SCALABLE, EFFICIENT, AND FAIR ALGORITHMS FOR STRUCTURED
CONVEX OPTIMIZATION PROBLEMS

A Dissertation
Presented to
The Academic Faculty

By

Mehrdad Ghadiri

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in
Algorithms, Combinatorics, and Optimization (ACO)
School of Computer Science
College of Computing

Georgia Institute of Technology
December 2023

© Mehrdad Ghadiri 2023
SCALABLE, EFFICIENT, AND FAIR ALGORITHMS FOR STRUCTURED
CONVEX OPTIMIZATION PROBLEMS

Thesis committee:

Dr. Santosh Vempala, Advisor
School of Computer Science
Georgia Institute of Technology

Dr. Swati Gupta
Sloan School of Management
Massachusetts Institute of Technology

Dr. Richard Peng
Computer Science Department
Carnegie Mellon University

Dr. Jan van den Brand, Reader
School of Computer Science
Georgia Institute of Technology

Dr. Mohit Singh
School of Industrial and Systems Engineering
Georgia Institute of Technology

Date approved: August 22, 2023
You ask me if an ordinary person—by studying hard—would get to be able to imagine these things like I imagine. Of course. I was an ordinary person who studied hard. There’s no miracle people. It just happens they got interested in this thing, and they learned all this stuff. They’re just people. There’s no talent or special miracle ability to understand quantum mechanics or a miracle ability to imagine electromagnetic fields that comes without practice and reading and learning and study. So if you take an ordinary person who’s willing to devote a great deal of time and study and work and thinking and mathematics, then he’s become a scientist.

Richard P. Feynman
To my family and all the past mathematicians and scientists who inspired me.
ACKNOWLEDGMENTS

It takes an army to train a mathematician, scientist, or researcher. I consider myself extremely fortunate to have had the opportunity to work with and learn from many. I am deeply grateful to my advisor, Santosh Vempala, who not only provided me with the opportunity and freedom to explore a diverse range of fundamental problems but also inspired me to tackle challenging research endeavors. His guidance has been instrumental in shaping my future research pursuits. I would also like to express my heartfelt appreciation to Richard Peng, who I consider a second advisor throughout my Ph.D. journey. Engaging in conversations with Richard about mathematics, research, academia, and life has been invaluable to my growth. The breakthrough work of Santosh and Richard on solving sparse linear systems was a pivotal moment in my Ph.D. and exposed me to a wealth of fascinating mathematical concepts.

Furthermore, I wish to convey my sincere appreciation to Mohit Singh. I have learned a great deal from him regarding the geometry of polytopes and linear inequalities. Our discussions on topics such as the traveling salesperson problem, $k$-means clustering, and iterative rounding have enriched my Ph.D. experience. Additionally, I have been fortunate to serve as a teaching assistant for Jan van den Brand in my final year at Georgia Tech and work with him on various problems. I have learned numerous techniques in dynamic algorithms from him. I am also grateful to Swati Gupta for agreeing to be on my committee and for the extensive knowledge I gained from her combinatorial optimization course.

I would like to extend my gratitude to Vahab Mirrokni, Matthew Fahrbach, and Gang Fu for the invaluable time I spent at Google, where I had the opportunity to work on fundamental tensor decomposition problems in an industrial setting. Their support and encouragement have been greatly appreciated. Furthermore, I am deeply thankful to Yin-Tat Lee and Zhao Song for facilitating my visit to the University of Washington in Seattle. I would also like to thank Anup Rao, Cameron Musco, Tung Mai, and David Arbour for giving
me the chance to contribute to the applications of numerical linear algebra in statistics. My
time at Adobe, under the guidance of exceptional individuals, allowed me to acquire invalu-
able knowledge about statistics and its real-world applications. I am especially grateful to
Samira Samadi for introducing me to the field of ethical optimization and fairness.

Last but certainly not least, I am immensely grateful to my family for their unwavering
support and encouragement. From my mom (Fatemeh) and dad (Morteza), I learned the
importance of hard work and determination, which ultimately led me to pursue a career in
mathematics and science. Their encouragement from an early age has been instrumental in
my journey. I would also like to express my heartfelt appreciation to my brother and sister,
Mohammad and Niloufar, for always being there for me.
TABLE OF CONTENTS

Acknowledgments ................................................................. v

List of Tables ................................................................. xiii

List of Figures ................................................................. xvi

Summary ................................................................. xx

Chapter 1: Introduction ...................................................... 1
  1.1 Stability and Bit Complexity Analysis of Algorithms .............. 3
  1.2 Tensor Decomposition ................................................. 11
  1.3 Fairness in Machine Learning ........................................ 17

Chapter 2: The Bit Complexity of Efficient Continuous Optimization .... 22
  2.1 Introduction .............................................................. 22
    2.1.1 Results ............................................................ 26
    2.1.2 Techniques ........................................................ 33
    2.1.3 Discussion ......................................................... 36
    2.1.4 Notation and Preliminaries .................................... 37
  2.2 Linear Regression ..................................................... 41
  2.3 Inverse Maintenance Stability ...................................... 48
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.3.1</td>
<td>Bit Complexity of Dense Inverse Maintenance Data Structure</td>
<td>52</td>
</tr>
<tr>
<td>2.3.2</td>
<td>Bit Complexity of Sparse Inverse Maintenance Data Structure</td>
<td>56</td>
</tr>
<tr>
<td>2.4</td>
<td>Linear Programming Using Interior Point Methods (IPM)</td>
<td>62</td>
</tr>
<tr>
<td>2.4.1</td>
<td>LP Preliminaries and Initial Feasible Point</td>
<td>64</td>
</tr>
<tr>
<td>2.4.2</td>
<td>Robust Interior Point Method For Solving Linear Programs</td>
<td>70</td>
</tr>
<tr>
<td>2.4.3</td>
<td>Inverse Free Path Following IPM</td>
<td>81</td>
</tr>
<tr>
<td>2.4.4</td>
<td>Solving Sparse LP faster than Matrix Multiplication for ( \omega &gt; 2.5 )</td>
<td>90</td>
</tr>
<tr>
<td>2.5</td>
<td>( p )-Norm Regression</td>
<td>97</td>
</tr>
<tr>
<td>2.5.1</td>
<td>Residual Problem</td>
<td>98</td>
</tr>
<tr>
<td>2.5.2</td>
<td>Solving The Residual Problem</td>
<td>103</td>
</tr>
<tr>
<td>2.5.3</td>
<td>Weighted Linear Regression with Equality Constraints</td>
<td>111</td>
</tr>
<tr>
<td>2.5.4</td>
<td>Mixed ((2, \infty))-Norm Minimization</td>
<td>115</td>
</tr>
<tr>
<td>2.5.5</td>
<td>Mixed ((2, p))-Norm Minimization</td>
<td>133</td>
</tr>
<tr>
<td>2.6</td>
<td>Low-Rank Matrices</td>
<td>145</td>
</tr>
</tbody>
</table>

**Chapter 3: Symmetric Factorizations of Hankel Matrices**

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Introduction</td>
<td>147</td>
</tr>
<tr>
<td>3.1.1</td>
<td>Notation and Preliminaries</td>
<td>151</td>
</tr>
<tr>
<td>3.2</td>
<td>Results and Conjectures</td>
<td>154</td>
</tr>
<tr>
<td>3.3</td>
<td>Motivation and Related Work</td>
<td>157</td>
</tr>
<tr>
<td>3.3.1</td>
<td>Faster Sparse Linear System Solvers for Batch Problems</td>
<td>157</td>
</tr>
<tr>
<td>3.3.2</td>
<td>Solving Linear Programs Faster than Matrix Multiplication</td>
<td>165</td>
</tr>
<tr>
<td>3.3.3</td>
<td>SoS decomposition of polynomials</td>
<td>168</td>
</tr>
<tr>
<td>Section</td>
<td>Title</td>
<td>Page</td>
</tr>
<tr>
<td>---------</td>
<td>-------</td>
<td>------</td>
</tr>
<tr>
<td>3.4</td>
<td>Symmetric Factorization of Hermitian Toeplitz Matrices</td>
<td>169</td>
</tr>
<tr>
<td>3.5</td>
<td>Key Identity for Hankel Matrices</td>
<td>173</td>
</tr>
<tr>
<td>3.6</td>
<td>Symmetric Factorization of Hankel Matrices</td>
<td>176</td>
</tr>
<tr>
<td>3.7</td>
<td>Symmetric Factorization of Inverses of Hankel Matrices</td>
<td>184</td>
</tr>
<tr>
<td>3.8</td>
<td>Discussion and Conclusion</td>
<td>192</td>
</tr>
<tr>
<td>4.1</td>
<td>Subquadratic Kronecker Regression and Applications to Tensor Decomposition</td>
<td>194</td>
</tr>
<tr>
<td>4.1.1</td>
<td>Introduction</td>
<td>194</td>
</tr>
<tr>
<td>4.1.2</td>
<td>Our Contributions and Techniques</td>
<td>197</td>
</tr>
<tr>
<td>4.1.2</td>
<td>Related Work</td>
<td>199</td>
</tr>
<tr>
<td>4.2</td>
<td>Preliminaries</td>
<td>200</td>
</tr>
<tr>
<td>4.3</td>
<td>Row Sampling and Approximate Regression</td>
<td>203</td>
</tr>
<tr>
<td>4.4</td>
<td>Kronecker Regression</td>
<td>206</td>
</tr>
<tr>
<td>4.4.1</td>
<td>Iterative Methods</td>
<td>209</td>
</tr>
<tr>
<td>4.4.2</td>
<td>Fast Kronecker-Matrix Multiplication</td>
<td>209</td>
</tr>
<tr>
<td>4.4.3</td>
<td>Main Algorithm</td>
<td>216</td>
</tr>
<tr>
<td>4.5</td>
<td>Applications to Low-Rank Tucker Decomposition</td>
<td>220</td>
</tr>
<tr>
<td>4.6</td>
<td>Experiments</td>
<td>229</td>
</tr>
<tr>
<td>4.6.1</td>
<td>Low-rank Tucker Decomposition of Image Tensors</td>
<td>231</td>
</tr>
<tr>
<td>4.7</td>
<td>Missing Analysis from section 4.3</td>
<td>234</td>
</tr>
<tr>
<td>4.7.1</td>
<td>Approximate Least Squares</td>
<td>235</td>
</tr>
<tr>
<td>4.7.2</td>
<td>Generalizing to Submatrix Sketching</td>
<td>241</td>
</tr>
</tbody>
</table>
### Chapter 6: Fair $k$-means Clustering

6.2.2 Computing Fair Centers via Line Search .......................... 289
6.2.3 Fair $k$-means is well-behaved ................................. 292

6.3 Generalization to $m > 2$ groups ................................. 295
6.3.1 Multiplicative Weights Update Heuristic ....................... 298
6.3.2 Certificate of Optimality ......................................... 299
6.3.3 Stability and Approximability .................................... 301

6.4 Experimental Evaluation ............................................. 301
6.5 Discussion ............................................................. 305

### Chapter 7: Scalable Constant-Factor Approximation Algorithms for Socially Fair $k$-Clustering

7.1 Introduction .......................................................... 308
7.1.1 Approach and Techniques ......................................... 312
7.1.2 Preliminaries ...................................................... 314
7.1.3 Related Work ...................................................... 314

7.2 Bicriteria Approximation ............................................. 316

7.3 Approximation Algorithms for Fair $k$-Clustering ................... 323
7.4 Converting a Solution with $k + m$ centers to one with $k$ centers .... 325
7.5 Speeding Up the Algorithm ........................................... 331

7.6 Empirical Study ....................................................... 335
7.6.1 Comparison of Algorithms Showing Both Maximum and Minimum 338
7.6.2 Results of Different Algorithms for Different Parameters .... 339
7.6.3 Comparison of Running Time of Different Algorithms in Practice 339
7.6.4 Empirical Coreset Results on Full-Size Datasets ............... 342
7.6.5 Number of Selected Centers in The Bicriteria Algorithms . . . . . . 343

7.7 Conclusion . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 343

Chapter 8: Open Problems and Future Research . . . . . . . . . . . . . . . . . . . 348

8.1 Stability and Bit Complexity of Efficient Algorithms for Optimization . . 348
8.2 Tensor Decompositions . . . . . . . . . . . . . . . . . . . . . . . . . . . . 350
8.3 Fair Clustering . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 352

References . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 353
LIST OF TABLES

3.1  Comparison of running times of [45] with an algorithm implied by our conjecture for solving a batch of linear systems for a poly-conditioned matrix with $O(n)$ nonzero entries. .................................................. 150

4.1  Running times of TuckerALS (Algorithm 16) factor matrix and core tensor updates for $\lambda = 0$ using different Kronecker regression methods. The factor matrices are denoted by $A^{(n)} \in \mathbb{R}^{I_n \times R_n}$. The input tensor has size $I = I_1 \cdots I_N$ and the core tensor has size $R = R_1 \cdots R_N$. Let $I_{\neq k} = I/I_k$ and $R_{\neq k} = R/R_k$. We use $\omega < 2.373$ for the matrix-multiplication exponent and the constant $\theta^* > 0$ for the optimally balanced fast rectangular matrix multiplication as stated in Theorem 4.4.2.4, i.e., $R^{2-\theta^*} = \min_{T \subseteq \{N\}} \text{MM} (\prod_{n \in T} R_n, R, \prod_{n \notin T} R_n)$. Factors of $N$ are dropped for notational brevity. ................................................................. 198

4.2  Kronecker regression losses for $d = 64$. OPT denotes the loss of the KronMatMul algorithm, DJSSW19 is [56, Algorithm 1], and Algorithm 17 is FastKroneckerRegression. We also record the relative error of each algorithm and the number of rows sampled from $A^{(1)} \otimes A^{(2)}$. ............. 231

4.3  Relative reconstruction errors for cardiac MRI tensor with different multilinear ranks. ................................................................. 232

4.4  Average iteration time of ALS with sketching-based Kronecker regression for cardiac MRI tensor with different multilinear ranks (seconds). .......... 232

4.5  Relative reconstruction errors for cardiac MRI tensor with multilinear rank $(4, 4, 2, 2)$ during ALS with and without using FastKroneckerRegression as a subroutine. ................................................................. 233

4.6  Relative reconstruction errors for cardiac MRI tensor with multilinear rank $(8, 8, 4, 4)$ during ALS with and without using FastKroneckerRegression as a subroutine. ................................................................. 233
4.7 Relative reconstruction errors for hyperspectral tensor with different multilinear ranks. .............................................. 233

4.8 Average iteration time of ALS with sketching-based Kronecker regression for the hyperspectral image tensor with different multilinear ranks (seconds). 234

4.9 Relative reconstruction errors for the COIL-100 tensor with different multilinear ranks. .............................................. 234

4.10 Average iteration time of ALS with sketching-based Kronecker regression for the COIL-100 tensor with different multilinear ranks (seconds). 234

5.1 Statistics for tensor datasets used in experiments. ................................. 276

7.1 Comparison of the running time of different algorithms on the first 200 samples of the Credit dataset averaged over five runs. ................................. 341

7.2 Comparison of the running time of different algorithms on the first 200 samples of the COMPAS dataset averaged over five runs. ................................. 341

7.3 Comparison of the running time of different algorithms on the first 200 samples of the Adult dataset with 5 race groups averaged over five runs. ................................. 341

7.4 Comparison of the running time of different algorithms on the first 500 samples of the Adult dataset with 10 race and gender groups averaged over five runs. ................................. 341

7.5 The number of selected centers for Abbasi-Bhaskara-Venkataramanan algorithm [74] (bottom table) and our bicriteria algorithm (top table) on the Credit dataset. $\lambda$ is a parameter of our algorithm and denotes the amount of decrease in radii of balls around the clients in the iterative rounding algorithm. $\epsilon$ is a parameter of ABV algorithm, and the maximum number of selected centers is $k/(1 - \epsilon)$, which achieves a $2/\epsilon$ approximation factor.. ................................. 344

7.6 The number of selected centers for Abbasi-Bhaskara-Venkataramanan algorithm [74] (right table) and our bicriteria algorithm (left table) on the COMPAS dataset. $\lambda$ is a parameter of our algorithm and denotes the amount of decrease in radii of balls around the clients in the iterative rounding algorithm. $\epsilon$ is a parameter of ABV algorithm, and the maximum number of selected centers is $k/(1 - \epsilon)$, which achieves a $2/\epsilon$ approximation factor. ................................. 345
7.7 The number of selected centers for Abbasi-Bhaskara-Venkatasubramanian algorithm [74] (right table) and our bicriteria algorithm (left table) on the Adult dataset with 5 groups. \( \lambda \) is a parameter of our algorithm and denotes the amount of decrease in radii of balls around the clients in the iterative rounding algorithm. \( \epsilon \) is a parameter of ABV algorithm, and the maximum number of selected centers is \( k/(1-\epsilon) \), which achieves a \( 2/\epsilon \) approximation factor. .......................................................... 346

7.8 The number of selected centers for Abbasi-Bhaskara-Venkatasubramanian algorithm [74] (bottom table) and our bicriteria algorithm (top table) on the Adult dataset with 10 groups. \( \lambda \) is a parameter of our algorithm and denotes the amount of decrease in radii of balls around the clients in the iterative rounding algorithm. \( \epsilon \) is a parameter of ABV algorithm, and the maximum number of selected centers is \( k/(1-\epsilon) \), which achieves a \( 2/\epsilon \) approximation factor. .......................................................... 347
LIST OF FIGURES

1.1 A visual example of the simplex method. The picture is from the Wikipedia entry for the simplex algorithm [29]. .................................................. 6

1.2 A visual example of the ellipsoid method. ......................................... 7

1.3 The running time (in terms of number of bit operations) of different sparse problems. For linear systems, \( p \)-norm regression, and linear programs, the mentioned running times are for \( \text{nnz}(A) = O(n) \). Moreover for \( p \)-norm regression, \( p \to \infty \). ................................................................. 10

1.4 Examples of applications of tensors. .................................................. 12

1.5 An example of the result of \( \mathcal{F} \times_1 A^{(1)} \) .................................. 13

1.6 An example of mode-1 unfolding of tensor \( \mathcal{T} \) ................................. 13

1.7 A visual example of Kronecker product of two matrices. ...................... 14

1.8 Pareto frontier of core shapes \( \mathbf{r} \in [20]^3 \) for hyperspectral tensor \( \mathcal{T} \in \mathbb{R}^{1024\times1344\times33} \) [58]. Plots the relative reconstruction error (RRE), i.e., \( L(\mathcal{T}, \mathbf{r}) / \|\mathcal{T}\|_F^2 \), as a function of the budget on the total size of the decomposition. RRE-greedy builds core shapes by computing Tucker decompositions at each step. HOSVD-IP is Algorithm 20 with integer programming, which builds core shapes via a surrogate packing problem on higher-order singular values. ................................................................. 16

1.9 The standard Lloyd’s algorithm results in a significant gap in the average clustering costs of different subgroups of the data. ......................... 18

1.10 Two demographic groups are shown with blue and purple. The 2-means objective minimizing the average clustering cost prefers the clustering (and centers) shown in the left figure. This clustering incurs a much higher average clustering cost for purple than for blue. The clustering in the right figure has more equitable clustering cost for the two groups. .................. 19
4.1 Running times of Kronecker regression algorithms with a design matrix of size $n^2 \times d^2$. ................................................................. 229

5.1 Pareto frontier of core shapes $r \in [20]^3$ for hyperspectral tensor $X \in \mathbb{R}^{1024 \times 1344 \times 33}$. Plots the RRE, i.e., $L(X, r)/\|X\|_F^2$, as a function of compression rate. RRE-greedy builds core shapes by computing Tucker decompositions at each step. HOSVD-IP is Algorithm 20 with integer programming, which builds core shapes via a surrogate packing problem on higher-order singular values. ................................................................. 249

5.2 Tree tensor network corresponding to a Tucker decomposition. ................. 273

5.3 Tree tensor network example corresponding to a hierarchical Tucker decomposition. ................................................................. 273

5.4 Comparison of five core shape solvers on four real-world tensors (columns) for increasing values of the Tucker decomposition size budget $c \leq 100,000$. The plots in the top row are the HOSVD Tucker packing objective value $f(r)$ for the core shape solutions $r$, the middle row is the RRE, and the bottom row is the running time of each algorithm in seconds. ................................................................. 275

6.1 The standard Lloyd’s algorithm results in a significant gap in the average clustering costs of different subgroups of the data. ....................... 281

6.2 Two demographic groups are shown with blue and purple. The 2-means objective minimizing the average clustering cost prefers the clustering (and centers) shown in the left figure. This clustering incurs a much higher average clustering cost for purple than for blue. The clustering in the right figure has more equitable clustering cost for the two groups. ....................... 284

6.3 Left: an example of the one-dimensional curve for $k = 2$. Right: the functions $f_A$ and $f_B$ with respect to $\gamma$, and two steps of the line search algorithm. We can use a line search to find the optimal value of $\gamma$ and an optimal solution to (Equation 6.2.3). ....................... 291

6.4 An example of $k$-means problem where the current clustering is not a local optimal and we need to check all the possible partitions with the current centers. $c_1, c_2, c_3$ are the centers and the points are marked with the letter A on top of them. ....................... 294
6.5 An example that shows $F$ is not quasiconvex. The yellow area represents the points for which the value of $F$ is less than 4.2. As one can see, the yellow area is not convex and therefore $F$ is not quasiconvex in terms of $\gamma_j$’s. The table shows the parameters that were used for this example.

6.6 Average clustering cost of different groups when using Fair-Lloyd algorithm versus the standard Lloyd’s. Rows correspond to different pre-processing methods and columns to the datasets. Note that the fair clustering costs for the two groups are identical or nearly identical in all datasets.

6.7 Adult dataset: The maximum ratio of average clustering cost between any two racial groups: “Amer-Indian-Eskim”, “Asian-Pac-Islander”, “Black”, “White”, and “Other”.

6.8 Adult dataset: comparison of the standard Lloyd’s and Fair-Lloyd algorithm for the three different pre-processing choices of w/o PCA, w/ PCA, and w/ Fair-PCA.

6.9 Running time (seconds) of Fair-Lloyd algorithm versus the standard Lloyd’s algorithm on the $k$-dimensional PCA space for 200 iterations.

6.10 Convergence rate of Fair-Lloyd algorithm versus the standard Lloyd’s algorithm for $k = 10$. The plotted objective value for the standard Lloyd is the average cost of clustering over the whole population, and the objective value for Fair-Lloyd is the maximum average cost of the demographic groups. The reported objective values are averaged over 20 runs and the shaded areas are the standard deviations.

6.11 Comparison of socially fair $k$-means (Fair-Lloyd) to proportionally fair $k$-means (Fairlet) on the Credit and Adult dataset in terms of proportionality and clustering cost.

7.1 (a) Distance of $i'$ from the facilities of its representative $i$. (b) Solid and dashed circles are the balls corresponding to representative ($\mathcal{U}^*$) and non-representative clients ($\mathcal{U}'$), respectively.

7.2 Comparison of our bicriteria algorithm with ABV [74]. The number of centers our algorithm selects is close to $k$ and is often smaller than ABV (see Section 7.6.5).

7.3 Comparison of our algorithm with $k$ centers with MV [75].
7.4 Comparison of our bicriteria algorithm with ABV [74]. The max and min on Subfigure (c) are across the demographic groups and are used to prevent cluttering plots with 5 groups. The number of centers our algorithm selects is close to $k$ and is often smaller than ABV (see Section 7.6.5). 338

7.5 Comparison of our algorithm with exactly $k$ centers with MV [75]. The max and min on Subfigure (c) are across the groups and are used to prevent cluttering plots with 5 groups. 339

7.6 Performance of our bicriteria algorithm for different values of $\lambda$. The max and min on Subfigure (c) are across the demographic groups. 339

7.7 Performance of our bicriteria algorithm of ABV [74] for different values of $\epsilon$. The max and min on Subfigure (c) are across the demographic groups. 340

7.8 Performance of our algorithm with exactly $k$ centers for different values of $\lambda$. The max and min on Subfigure (c) are across the demographic groups. 340

7.9 Performance of the MV algorithm [75] for different values of $\gamma$. The max and min on Subfigure (c) are across the demographic groups. 340

7.10 Results of the Credit dataset with 2 groups and 30,000 datapoints using the approach of Section 7.5. 342

7.11 Results of the Adult dataset with 5 groups and 48,842 datapoints using the approach of Section 7.5. 343
SUMMARY

The growth of machine learning and data science has necessitated the development of provably fast and scalable algorithms that incorporate ethical requirements. In this thesis, we present algorithms for fundamental optimization algorithms with theoretical guarantees on approximation quality and running time.

We analyze the bit complexity and stability of efficient algorithms for problems including linear regression, $p$-norm regression, and linear programming by showing that a common subroutine, inverse maintenance, is backward stable and that iterative approaches for solving constrained weighted regression problems can be carried out with bounded-error pre-conditioners. We also present conjectures regarding the running time of computing symmetric factorizations for Hankel matrices that imply faster-than-matrix-multiplication time algorithms for solving sparse poly-conditioned linear programs.

We present the first subquadratic algorithm for solving the Kronecker regression problem, which improves the running time of all steps of the alternating least squares algorithm for the Tucker decomposition of tensors. In addition, we introduce the Tucker packing problem for computing an approximately optimal core shape for the Tucker decomposition problem. We prove this problem is NP-hard and provide polynomial-time approximation schemes for it.

Finally, we show that the popular $k$-means clustering algorithm (Lloyd’s heuristic) can result in outcomes that are unfavorable to subgroups of data. We introduce the socially fair $k$-means problem for which we provide a very efficient and practical heuristic. For the more general problem of $(\ell_p, k)$-clustering problem, we provide bicriteria constant-factor approximation algorithms. Many of our algorithms improve the state-of-the-art in practice.
CHAPTER 1
INTRODUCTION

This thesis focuses on developing efficient and scalable algorithms for classic and modern optimization problems. Although polynomial-time algorithms are established and known for many of the considered problems, more efficient and scalable algorithms for these problems have been an active area of research in recent years. This is due to the rapid growth of very large datasets arising from real-world applications ranging from biology [1] to social networks [2]. In addition, the growing applications of machine learning and data science have introduced new optimization challenges arising from societal and ethical considerations such as fairness [3] and differential privacy [4] that are even required by law (to some extent) in some applications. These requirements and challenges have necessitated modern approaches and techniques in optimization algorithms.

Although there are general techniques for solving convex optimization problems [5], one can often exploit structures in a certain problem to develop fast and scalable algorithms while incorporating the extra constraints imposed by ethical requirements. For example, in Chapter 4, we exploit a certain structure of matrices involved in tensor decomposition to develop faster algorithms. Another example is the structure of a convex optimization problem for computing a fair set of centers for the socially fair $k$-means problem that we discuss in Chapter 6.

Our results in this thesis fall under three categories.

**Stability and bit complexity analysis of algorithms related to optimization** (Chapters 2 and 3). Many of the recent advances in improving the efficiency of (approximately) solving fundamental optimization problems (such as linear programming and $p$-norm regression problems) have been made under the exact arithmetic model (real-RAM) [6, 7,
To show that these algorithms are implementable using a Turing machine, one needs to show that the numbers produced in the course of the algorithm have a bounded bit complexity, i.e., the number of bits. In turn, the bounds on the bit complexity of the numbers allow us to bound the actual running time of algorithms, i.e., the running time under the word-RAM model or a Turing machine. Although for many optimization algorithms in the literature, it has already been discussed that one can use approximate steps, in the recent works that use modern techniques, the required bit complexity to guarantee the required approximations has not been addressed. The latter is essentially a numerical analysis and a numerical linear algebraic concern.

We address this for the state-of-the-art algorithms for solving linear programming and $p$-norm regression problems. To this end, a key result we develop is the backward-stability analysis of inverse maintenance techniques which have many applications across optimization and dynamic algorithms. We discuss the background and our results and technique in more depth in Section 1.1.

**Tensor decomposition and related problems** (Chapters 4 and 5). Tensors and tensor decompositions have a broad set of applications across sciences ranging from functional magnetic resonance imaging (fMRI) [9] to learning mixtures of Gaussian distributions [10]. Similar to low-rank matrix factorizations, the goal of tensor decompositions is to find a small representation of a tensor while preserving as much information as possible. We consider the Tucker decomposition of tensors, one of the two widely used tensor decompositions. We present more efficient algorithms for computing the low-rank Tucker decomposition of a tensor. In addition, we introduce the problem of low Tucker-rank decomposition subject to a memory constraint and we present fast algorithms for approximately solving this problem. An in-depth discussion is included in Section 1.2.

**Fair Clustering** (Chapters 6 and 7). Fairness in machine learning is an important societal consideration when developing algorithms for human-centric applications [11].
general notion of group fairness seeks to provide solutions with an equal average cost for demographic groups [12]. We present such a group fairness criterion for unsupervised learning. More specifically, we present an objective function for $k$-clustering problems that promotes a more equitable average cost for demographic groups. We call this objective function and the associated problem the socially fair $k$-clustering. In addition, we design a very efficient and practical algorithm for the socially fair $k$-means problem by adapting the Lloyd heuristic [13]. A detailed discussion is presented in Section 1.3.

In the remaining sections of this chapter, we delve into the history, background, motivation, and related work regarding these three topics. In addition, we give high-level views of the challenges, our proposed approaches, and our results. The rest of the thesis is organized as the following. We discuss problems related to stability and bit complexity of optimization algorithms in Chapters 2 and 3. We present our results on tensor decompositions in Chapters 4 and 5. We discuss fair clustering problems in Chapters 6 and 7. We finally discuss open problems and future directions of research regarding all of these topics in Chapter 8.

1.1 Stability and Bit Complexity Analysis of Algorithms

Although initially, computer scientists have considered polynomial time algorithms for many optimizations and algorithmic problems (to some extent due to P versus NP question), with the increasing applications of computer science and algorithms, the area of fast algorithms with the goal of designing more efficient algorithms has received significant attention. Some notable examples with significant improvement over trivial polynomial time algorithms are matrix multiplication [14, 15, 16], solving linear systems and linear regression problems [17, 18, 19], linear programming [20, 21, 22, 6, 7], $p$-norm regression [8], and network flow problems [23]. We note that even finding the first polynomial time algorithm for some of these problems has been very nontrivial, e.g., ellipsoid algorithms for solving linear programs.
An important consideration in the area of fast algorithms is the number system used for presenting the numbers involved in the algorithm. This is to determine the strength of the machine that runs the algorithm. For example, the so-called real-RAM model (that resembles the Blum–Shub–Smale machine [24]) allows for exact arithmetic (i.e., infinite precision) in constant time for addition, subtraction, multiplication, division, and comparison. If we allow rounding to integers and modulus operations, then one can solve PSPACE-complete problems in polynomial time [25]. Therefore it is crucial to determine what operations are allowed and the running time of each operation. For example, in the real-RAM model, it is assumed that each arithmetic operation takes only a constant time. Note that such a running time is not possible if we implement our algorithm on a Turing machine. For example, if we use the binary representation of numbers even to represent (and read/write) an integer \( n \) on a Turing machine, we need \( \Omega(\log n) \) time. The other important aspect is the output we receive after performing the arithmetic operations. For example, in the widely used floating-point model, the output is only within a \((1 + \epsilon)\) multiplicative factor of the true result. Another example is for the rational representation of numbers: does the output give a coprime pair of numerator and denominator or not? This is important since it can affect how large the numbers in the algorithm can become, which affects the algorithm’s running time. Chapter 2 of [5] gives an in-depth discussion, especially on rational numbers.

In Chapters 2 and 3, we present our algorithms under fixed-point arithmetic in the word-RAM model. In particular, our numbers use a fixed number of bits before and after the decimal point, for example, (Equation 1.1.1) illustrates a fixed-point number with seven bits.

\[
1 0 1 0 . 0 1 1
\]

The number of bits used for the numbers depends on the problem we want to solve, for
example, it depends on the logarithm of the reciprocal of the error parameter (set by the input). For numbers with a total of $\ell$ bits before and after the decimal point, we can do the following operations in $O(\ell)$ time: addition, subtraction, and comparison. Multiplication of two such numbers can be done in $O(\ell \log \ell)$ using fast Fourier transform (FFT) — see Chapter 30 of [26]. Moreover, Under fixed-point arithmetic, our errors are additive (as opposed to multiplicative errors for floating-point arithmetic), i.e., the difference between the true number and what we store/compute is at most $\epsilon$. Therefore we can divide two numbers and find the square root of a number in $O(\ell \log(\ell) \cdot \log(1/\epsilon))$ such that the result is only additively away from the true answer by $\epsilon$. In Section 2.4.3, we use a result of Storjohann to solve integer linear systems of the form $Ax = b$ that output the solution as a rational vector where the denominator of entries is the determinant of $A$ [18]. We make sure to convert from/to fixed-point numbers before and after using Storjohann’s algorithm.

A very fundamental problem in optimization is linear programming. As Dantzig [27] puts it, “Linear Programming is viewed as a revolutionary development, giving us the ability for the first time to state general objectives and to find, by means of the simplex method, optimal policy decisions to practical decision problems of great complexity.”

In its standard form, a linear programming (LP) problem asks for a vector $x$ that

\begin{align}
\text{maximize } & \mathbf{c}^T \mathbf{x}, \\
\text{subject to } & A \mathbf{x} \leq \mathbf{b}, \\
& \mathbf{x} \geq 0,
\end{align}

where $A \in \mathbb{R}^{n \times d}$, $\mathbf{b} \in \mathbb{R}^n$, and $\mathbf{c} \in \mathbb{R}^d$. The first algorithm, called the simplex method, for solving such problems was developed more than seven decades ago [28]. The constraints $A \mathbf{x} \leq \mathbf{b}$ and $\mathbf{x} \geq 0$ define a geometric shape called a polyhedron (which is the intersection of a set of half-spaces). A vertex of the polyhedron is the intersection of $d$ of the hyper-planes (associated with constraints) with linearly independent vector normals. Two vertices
are neighbors if they share \( d - 1 \) hyperplanes. The simplex method starts from a vertex of the polyhedron and continues by hopping onto the neighboring vertices until it reaches a vertex that maximizes the objective function \( \mathbf{c}^\top \mathbf{x} \) — see Fig. 1.1 for a visual example. The pivoting rule determines to which neighboring vertex the algorithm should jump. Although the polynomial Hirsch conjecture states that the diameter of the graph of vertices is a polynomial in \( n \) and \( d \), no known pivoting rule finds the solution in a polynomial number of steps.

Khachiyan presented the first polynomial-time algorithm for linear programming in 1979, called the ellipsoid method [20]. Although the ellipsoid method is theoretically more efficient, the simplex method is more efficient in practice. This caused discussions in the complexity theory community about the value of polynomial running times for algorithms and how to measure encoding lengths and running times. A related discussion is the existence of strongly polynomial time algorithms for solving LPs, which is still an open problem. A strongly polynomial time algorithm is an algorithm that uses polynomial space and performs a number of elementary arithmetic operations which is bounded by a polynomial in the number of input numbers. In other words, the number of arithmetic operations of the
algorithm does not depend on the size and bit complexity of input numbers.

The version of the ellipsoid method that checks whether a polytope (a bounded polyhedron) is nonempty starts from a trivial ellipsoid that contains the polytope (a ball with the center at the origin). Then in each iteration, it asks an oracle whether the center of the ellipsoid is in the polytope or for a hyperplane that separates the center of the ellipsoid from the polytope. Then it computes a new ellipsoid that contains the intersection of the old ellipsoid and the half-space that contains the polytope — see Fig. 1.2 for a visual example. The convergence of the algorithm is derived from a volume argument that shows that the volume of the new ellipsoid is smaller than the volume of the old ellipsoid. Compared to the simplex method, an advantage of the ellipsoid method is that it does not need explicit access to the formulation of (Equation 1.1.2) and only a separation oracle that outputs a hyperplane that separates a point from the polytope is sufficient [30]. This makes the ellipsoid method particularly useful for combinatorial optimization problems where the size of the matrix can be exponential, for example, in terms of the number of vertices of a graph.

Shortly after the introduction of the ellipsoid method, Karmarkar [21] presented another polynomial time algorithm for linear programming called the interior point method (IPM). The IPM approach starts from a feasible point in the interior of the polytope and, in each iteration, produces another feasible point in the interior. Each solution is better than the previous one with respect to a criterion called duality gap, e.g., the duality gap of the
optimal solution is zero. The IPM approach is very efficient both in theory and practice. Each iteration of the IPM approach requires the solution to a system of linear equations. This emphasizes the close connection between linear algebra (more specifically, solving linear systems) and linear programming. The idea of improving the running time of solving linear systems to achieve improved running times for solving linear programs has been considered a viable approach since the early days of IPMs. Indeed even the original paper of Karmarkar discusses the idea of updating the inverse of a matrix that goes through a rank-one update which is called inverse maintenance — see Fact 2.1.2.1. Essentially for two linear systems $A^{(1)}x = b^{(1)}$ and $A^{(2)}x = b^{(2)}$ with rank$(A^{(2)} - A^{(1)}) = 1$, we can find the solution of the first one by computing the inverse of $A^{-1}$ and computing $(A^{(1)})^{-1}b$ in $O(n^2)$ arithmetic operations. Then the inverse of $A^{(2)}$ can be computed from $(A^{(1)})^{-1}$ in $O(n^2)$ arithmetic operations, and the second linear system can be solved with only $O(n^2)$ arithmetic operations. By employing this idea and fast matrix multiplication algorithms, Vaidya improved the running time of the Karmarkar algorithm from $O(n^{3.5})$ arithmetic operations (the trivial implementation) to $O(n^{2.5})$ arithmetic operations [31, 22]. He also discussed the bit complexity of numbers involved in the algorithm and concluded that it is sufficient to work with numbers with $L := \log(1 + \text{det}_{\text{max}} + \|b\|_{\infty} + \|c\|_{\infty})$ bits, where $\text{det}_{\text{max}}$ is the largest absolute value of the determinant of square submatrices of $A$. Note that for random matrices, $\text{det}_{\text{max}}$ is exponential in $n$ with high probability [32]. This means that the running time of Vaidya’s algorithm is $O(n^{3.5})$ in the worst case under fixed-point arithmetic. Later works improved the running time of solving linear programs to $O(n^\omega)$ arithmetic operations [33, 7], where $\omega \approx 2.372$ is the current matrix multiplication exponent (i.e., two $n$-by-$n$ matrices can be multiplied in $O(n^\omega)$ arithmetic operations).

Although the ellipsoid method stimulated numerous research to accelerate the algorithm and make it more stable for numerical purposes [34, 35], it appears that the focus on the IPMs has mostly been on improving the number of arithmetic operations, especially in the recent works [33, 7] despite the superior performance of the IPMs for solving linear
programs in practice. This has been the case for other fundamental optimization problems that have been studied in computer science recently, including linear regression \[36\] (i.e., \(\min_x \|Ax - b\|_2\)), and \(p\)-norm regression problems \[8\] of the following form

\[
\text{maximize } \|x\|_p^p, \\
\text{subject to } Ax = b,
\]

In Chapter 2 (joint work with Richard Peng and Santosh Vempala \[37\]), we consider the bit complexity of efficient algorithms for basic problems in optimization, including \(p\)-norm regression and linear programming. For these problems, the state-of-the-art algorithms match the current time complexity of multiplying two \(n\)-by-\(n\) matrices in terms of the number of arithmetic operations (up to polylogarithmic factors). However, these results assume infinite precision arithmetic, and due to complicated inverse maintenance techniques, the actual running times of these algorithms are not known. We settle these algorithms’ running time and bit complexity by showing that a core common subroutine, *inverse maintenance*, is backward-stable and that iterative approaches for solving constrained weighted regression problems can be carried out with bounded-error pre-conditioners. We settle the bit complexity of robust interior-point method (IPM) for linear programming \[6, 7\], iterative refinement for \(p\)-norm regression \[8, 38\], and input-sparsity time algorithms for linear regression \[39, 40\] on general matrices. In particular, in Chapter 2, we show

1. linear programs can be solved using IPMs approximately in matrix multiplication time multiplied by polylog factors that depend on the condition number \(\kappa\) (i.e., the ratio of the largest to smallest singular value) of the matrix and the inner and outer radius of the LP;

2. the \(p\)-norm regression problem (for fixed \(p\)) can be solved approximately in matrix multiplication time multiplied by polylog factors in \(\kappa\);
Figure 1.3: The running time (in terms of number of bit operations) of different sparse problems. For linear systems, $p$-norm regression, and linear programs, the mentioned running times are for $\text{nnz}(A) = O(n)$. Moreover for $p$-norm regression, $p \to \infty$.

3. and linear regression can be solved approximately in input-sparsity time multiplied by polylog factors in $\kappa$.

As we mentioned, the idea of improving the running time of optimization algorithms by developing more efficient algorithms for solving linear systems has been around since the early days of IPMs. Since for some structured matrices, the linear systems can be solved faster than matrix multiplication time $O(n^\omega)$, this can lead to drastic time improvements for problems involving such structures. This has probably been best exemplified by near-linear time Laplacian solvers that have led to improved running times in many graph problems [41, 42, 43, 44].

A recent breakthrough by Peng and Vempala [45] has shown that for polyconditioned (i.e., the ratio of largest to smallest singular value is bounded by a polynomial in the size of the matrix) sparse matrices (with $o(n^\omega - 1)$ nonzero entries), a linear system can be solved faster than matrix multiplication time. This has created the expectation that perhaps $p$-norm regression and linear programming problems involving such matrices could also be solved faster than matrix multiplication time. In Sections 2.4.4 and 2.5, we discuss such approaches. By utilizing sparse linear system solver of Peng and Vempala, we show that $p$-norm regression problems on polyconditioned sparse matrices can be solved faster than $n^\omega$ for the current value of $\omega \approx 2.372$. For linear programs, we fall short and only show such results for $\omega > 2.5$. Fig. 1.3 denotes the current status of sparse problems — poly log factors are omitted in this picture.

In Chapter 3, we present conjectures regarding the running time of computing symmet-
ric factorizations of the form $BB^*$ for a certain class of structured matrices called Hankel matrices and their inverses under fixed-point arithmetic. An $n \times n$ matrix $H$ is called Hankel if for any $i, j, i', j' \in [n]$ with, $i + j = i' + j'$, $H_{ij} = H_{i'j'}$ — see (Equation 1.1.3) for an example. Hankel matrices are closely related to polynomials of single variables. With similar FFT techniques that allow almost linear time multiplication of two polynomials, a Hankel matrix can be applied to a vector in almost linear time.

$$H = \begin{bmatrix} h_1 & h_2 & h_3 & h_4 \\ h_2 & h_3 & h_4 & h_5 \\ h_3 & h_4 & h_5 & h_6 \\ h_4 & h_5 & h_6 & h_7 \end{bmatrix}.$$  \hfill (1.1.3)

If solved, these conjectures would result in a faster-than-matrix-multiplication algorithm for solving sparse poly-conditioned linear programming problems, a fundamental problem in optimization and theoretical computer science. In addition, these conjectures imply faster algorithms for solving a batch of sparse poly-conditioned linear systems compared to previous work [45, 46]. To justify our proposed conjectures and running times, we show weaker results of computing decompositions of the form $BB^* - CC^*$ for Hankel matrices and their inverses with the same running time. This chapter is based on my single-author paper [47].

1.2 Tensor Decomposition

Tensors are multidimensional arrays of numbers\(^1\). Vectors and matrices are tensors of order one and two, and the tensors of orders larger than two are called higher-order tensors. Tensors have a rich history in mathematics, physics, and natural sciences, where they

\(^1\)One can view them as linear operators as well, in which case each dimension can be considered to have infinitely many rows. However, in this work, we consider finite-size tensors.
first arose in the works of Gauss, studying differential geometry [48]. Other examples of tensors include functional magnetic resonance imaging (fMRI), representation of regular hypergraphs \(^2\), and knowledge bases — see Fig. 1.4 for these applications.

The goal of tensor decompositions, similar to low-rank matrix factorizations, is to compress the information represented in tensors in smaller objects, i.e., smaller matrices and tensors. This might be with the goal of determining the most important directions (e.g., principal component analysis) or to just simply decrease the size of the objects (e.g., to fit on a bounded memory). Perhaps the two most used tensor decompositions are canonical polyadic (CP) decomposition (also called tensor rank decomposition) [52], and Tucker decomposition [53]. The (low-rank) Tucker decomposition writes a tensor \( T \in \mathbb{R}^{I_1 \times \cdots \times I_N} \) of order \( N \) as the product of a core tensor \( G \in \mathbb{R}^{I_1 \times \cdots \times I_N} \) and \( N \) factor matrices \( A_1 \in \mathbb{R}^{I_1 \times R_1}, \ldots, A_N \in \mathbb{R}^{I_N \times R_N} \), where the goal is to find \( G, A_1, \ldots, A_N \) that

\[
\text{minimize } \| T - G \times_1 A_1 \times_2 A_2 \cdots \times_N A_N \|_F^2, \quad (1.2.1)
\]

where \( \| \cdot \|_F^2 \) is the sum of the squared value of the entries of the corresponding tensor (\( \| \cdot \|_F \) is called the Frobenius norm). \( \times_n \) are called mode-\( n \) products and are essentially the result

\(^2\)Each hyperedge corresponds to an entry of the tensor and each dimension of the tensor specifies one node of a hyperedge.
of splitting a tensor in dimension $n$, multiplying it by $A_n$ and concatenating the results — see Fig. 1.5 for a visual example for $G \times_1 A_1$. Note that mode-$n$ products commute for different values of $n$ and the size of dimensions of $G \times_1 A_1 \times_2 A_2 \cdots \times_N A_N$ is exactly equal to that of $T$.

The CP decomposition is a special case of the Tucker decomposition in which the core tensor $G$ is restricted to be diagonal. A noteworthy application of the CP decomposition (among many) is the study of learning a mixture of Gaussian distributions from empirical moments [54, 10]. Both CP and Tucker decompositions can be considered as generalizations of the singular value decomposition (SVD) to higher-order tensors. However, the phrase higher-order SVD (HOSVD) is usually only used for Tucker decomposition because of its close connections to the SVDs of matricizations of the tensor. Indeed to have unitary factor matrices, one can set them equal to left singular vectors of matricizations of the tensor [55]. The mode-$n$ matricization (also called unfolding) of the tensor is to slice the tensor and concatenate it into a matrix so that the columns of the resulting matrix are along the $n$-dimension of the tensor — see Fig. 1.6 for a visual example.

The alternating least squares (ALS) algorithm is a heuristic approach for solving the Tucker decomposition (Equation 1.2.1). In each iteration of this algorithm, all but one
of \( G, A_1, \ldots, A_N \) is fixed, and the problem is optimized with respect to the unfixed object. The algorithm alternates the unfixed object between \( G, A_1, \ldots, A_N \) until convergence. This heuristic is very fast in practice and usually only needs a few iterations to converge. The iterations of the ALS algorithm for Tucker decomposition is closely related to a class of structured matrices that arise from Kronecker product of matrices. The Kronecker product of two matrices \( A_1 \in \mathbb{R}^{I_1 \times R_1} \) and \( A_2 \in \mathbb{R}^{I_2 \times R_2} \) denoted by \( A_1 \otimes A_2 \) is a matrix in \( \mathbb{R}^{(I_1 I_2) \times (R_1 R_2)} \), where all the entries of \( A_1 \) is multiplied by all entries of \( A_2 \) to obtain the entries of \( A_1 \otimes A_2 \) — see Fig. 1.7 for a visual example. Indeed the iterations corresponding to fixing \( A_1, \ldots, A_N \), correspond to solving a linear regression problem where the matrix is \( A_1 \otimes \cdots \otimes A_N \). In Section 4.5, we discuss how other iterations of the ALS algorithm also correspond to linear regression problems involving Kronecker products.

For solving a linear regression problem of the form \( \min_x \| (A_1 \otimes \cdots \otimes A_N)x - b \|_2^2 \) to a multiplicative \((1 + \epsilon)\) error, previous studies from the sampling/sketching literature have proposed algorithms with the following running times (in terms of number of arithmetic operations) using leverage score sampling for \( A^{(n)} \in \mathbb{R}^{I_n \times R_n} \) with \( I_n \geq R_n \), for all \( n \in [N] \), where \( R = \prod_{n=1}^{N} R_n \) and \( \omega < 2.373 \) denotes the matrix multiplication exponent [16]:

1. \( \tilde{O}(\sum_{n=1}^{N} (\text{nnz}(A^{(n)}) + R_n^\omega + R^\omega \epsilon^{-1})) \) by sampling \( \tilde{O}(R \epsilon^{-1}) \) rows of \( K \) by their leverage scores [56].

2. \( \tilde{O}(\sum_{n=1}^{N} (\text{nnz}(A^{(n)}) + R_n^\omega \epsilon^{-1}) + R^{-1}) \) by sampling \( \tilde{O}(R_n \epsilon^{-1}) \) rows from each factor matrix \( A^{(n)} \) and taking the Kronecker product of the sampled factor matrices [56].

In Chapter 4, we present the first subquadratic algorithm for the Kronecker regression

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>( \otimes )</th>
<th>4</th>
<th>3</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>1</td>
<td>=</td>
<td>5</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>24</td>
<td>18</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>30</td>
<td>12</td>
<td>18</td>
</tr>
</tbody>
</table>

Figure 1.7: A visual example of Kronecker product of two matrices.
problem and show that this can be applied to all of the steps of the alternating least squares (ALS) algorithm for Tucker decomposition of tensors. Our algorithm gives a significant improvement in the running time of solving this problem both in theory and practice. The theoretical improvement is from a running time of $O(R^\omega)$ (where $\omega \approx 2.373$ is the matrix multiplication exponent) to $R^{1.626}$, and the practical improvement is from $R^3$ to $R^2$ (since fast matrix multiplication is not used in practice). Our experiments show a 100x improvement in the running time of the ALS algorithm, even on relatively small datasets, and this improvement increases for larger datasets. Chapter 4 is based on a joint work with Matthew Fahrbach and Gang Fu [57].

Another important question regarding Tucker decomposition that puzzles practitioners is how to select the dimensions $R_1, \ldots, R_N$ of the core tensor $\mathcal{T}$. This is especially important when there is a memory constraint for the size of the decomposition. The total size of a Tucker decomposition with a core tensor $\mathcal{G} \in \mathbb{R}^{R_1 \times \cdots \times R_N}$ is

$$
\sum_{n=1}^{N} I_n R_n + \prod_{n=1}^{N} R_n. \quad (1.2.2)
$$

Then a memory budget constraint asks for a core-tensor shape (i.e., dimension sizes) such that (Equation 1.2.2) is less than a budget $B > 0$. To illustrate that this problem is important in practice, Fig. 1.8 shows the relative reconstruction error (RRE) defined as the following for different shapes $\mathbf{r} = (R_1, \ldots, R_N)$,

$$
\frac{L(\mathcal{T}, \mathbf{r})}{\|\mathcal{T}\|_F^2},
$$

where

$$
L(\mathcal{T}, \mathbf{r}) = \min_{\mathcal{G} \in \mathbb{R}^{I_1 \times \cdots \times I_N}, A_1 \in \mathbb{R}^{I_1 \times R_1}, \ldots, A_N \in \mathbb{R}^{I_N \times R_N}} \|\mathcal{T} - \mathcal{G} \times_1 A_1 \times_2 A_2 \cdots \times_N A_N\|_F^2.
$$

Note that for a certain budget, different shapes achieve different errors, and a practi-
Figure 1.8: Pareto frontier of core shapes $r \in [20]^3$ for hyperspectral tensor $\mathcal{T} \in \mathbb{R}^{1024 \times 1344 \times 33}$ [58]. Plots the relative reconstruction error (RRE), i.e., $L(\mathcal{T}, r)/\|\mathcal{T}\|_F^2$, as a function of the budget on the total size of the decomposition. RRE-greedy builds core shapes by computing Tucker decompositions at each step. HOSVD-IP is Algorithm 20 with integer programming, which builds core shapes via a surrogate packing problem on higher-order singular values.

A person wants to pick a shape on the Pareto frontier that matches their memory budget. In Chapter 5, we formulate this as a combinatorial optimization problem that aims to find an optimal core tensor shape, also called multilinear rank, for a size-constrained Tucker decomposition. This problem has many applications, including reducing the size of trained neural networks for usage on mobile phones [59]. There are two main challenges in solving such a combinatorial optimization problem.

1. We cannot evaluate the function $L(\mathcal{T}, r)$ since low-rank tensor decomposition is an NP-hard problem [60].

2. The constraint $\sum_{n=1}^N I_n R_n + \prod_{n=1}^N R_n \leq B$ is not a convex constraint.

To alleviate the first issue, we instead use a surrogate (loss) function that arises from singular values of matricization of the tensor. We call the optimization problem over the
surrogate loss the *Tucker packing problem*. We show that even the Tucker packing problem is NP-hard. However, we present a polynomial-time approximation scheme (PTAS) for this problem and show that one can bound the RRE by solving the Tucker packing problem approximately. Our experiments (implemented using an integer linear programming solver) show that the solution quality of our algorithm is competitive with (and sometimes better than) the greedy heuristics used by practitioners [61] that use the true Tucker decomposition loss at each step, while also running up to 1000x faster. Chapter 5 is based on a joint work with Matthew Fahrbach, Gang Fu, and Vahab Mirrokni [62].

### 1.3 Fairness in Machine Learning

In machine learning, group fairness is a popular approach to prevent unfavorable treatment of individuals based on sensitive attributes such as race, gender, and disability. This is a legal requirement by the Civil Rights Act of 1968 of the United States [63, 64]. In machine learning applications, this is usually interpreted as the average result for the demographic groups (e.g., race and gender groups). For example, for (binary) classification, one might want to have the same false positive rate for all gender groups [65]. The study of group fairness and fair algorithms became a center of attention due to catastrophic failure and biases of machine learning tools toward minority groups (such as African Americans) on a wide range of tasks from facial recognition to recidivism prediction [66, 67, 68, 69].

Although individual notions of fairness have also been introduced [12] and there are some critiques on using race and gender for making decisions (e.g., using race and gender in hiring decisions has resulted in lawsuits [70]), group fairness has remained an important topic of study in machine learning.

In unsupervised learning tasks (such as clustering or dimensionality reduction), group fairness usually tries to minimize the difference between the average loss (objective function) for demographic groups. For example, Samadi et al. [66] was the first to show that the average reconstruction error of the principal component analysis (PCA) is different for
different gender groups if classical algorithms that do not incorporate fairness criteria are used. They introduced dimensionality reduction algorithms that achieve equitable reconstruction errors for different demographic groups [66, 71].

For clustering problems that select a set of centers, the loss of each individual can be defined based on their distance to the closest center. Then the average loss for each demographic group can be defined based on the loss of individuals. For the popular $k$-means clustering, the loss of an individual is the distance (to the closest center) squared. The goal of $k$-means clustering is to select a set of centers of size $k$ that minimizes the average loss over the whole population.

Our experiments (illustrated in Fig. 1.9) show that the classic $k$-means objective and algorithm, which is one of the main tools used in machine learning for clustering, is not fair to different demographic groups of the population in terms of average clustering cost in experiments performed on benchmark datasets. To resolve this issue, we define a new objective function called socially fair $k$-means that minimizes the maximum average cost over the demographic groups. Fig. 1.10 illustrates that our objective function leads to more equitable average clustering costs across different demographic groups. This is also supported by our experiments on real-world datasets presented in Section 6.4.

Lloyd’s heuristic is the main approach for solving the $k$-means clustering problem. It starts from a random set of centers and then it repeats the following two-step process until convergence.
1. Form the clusters by connecting each client to the nearest center.

2. Update the set of centers to the mean (i.e., average) of clusters.

We design an algorithm called fair Lloyd, which is similar to Lloyd’s heuristic. The difference is that in the second step of the iterative process, we find a fair set of centers given the current clustering, where a fair set of centers is a set of centers that minimizes the maximum average clustering cost across the demographic groups. We show that this step can be performed by solving a constrained quadratic program. We present an algorithm to solve such problems very efficiently (linear in $k$). To analyze our algorithm, we use the properties of subgradients of the maximum of convex functions to rapidly find a set of centers with an equitable cost for different demographic groups.

Our experiments show that the running time of our algorithm is comparable to that of the popular Lloyd’s algorithm used for the classic $k$-means problem. Hence our algorithm can be used in any human-centric application where currently the $k$-means algorithm is used. Chapter 6 is based on a joint work with Samira Samadi and Santosh Vempala [72].

In Chapter 7 (based on a joint work with Mohit Singh and Santosh Vempala [73]), we

Figure 1.10: Two demographic groups are shown with blue and purple. The 2-means objective minimizing the average clustering cost prefers the clustering (and centers) shown in the left figure. This clustering incurs a much higher average clustering cost for purple than for blue. The clustering in the right figure has more equitable clustering cost for the two groups.
consider the more general problem of socially fair \((\ell_p, k)\)-clustering, which is a general-
ization of fair \(k\)-means. In the case of \(p = 2\), this problem is equivalent to the socially
fair \(k\)-means problem. For a candidate set of centers \(F\) and demographic groups of clients
\(A_1, \ldots, A_m\) and a metric \(d\) on the points of the problem, the fair \((\ell_p, k)\)-clustering problem
asks for the solution of the following, where

\[
\min_{F \subseteq \mathcal{F} : |F| = k} \max_{s \in [m]} \sum_{i \in A_s} w_s(i) d(i, F)^p, \\
\]

\(d(i, F)\) is the distance of the closest center in \(F\) to \(i\), and \(w_s\) is a weight vector on the
members of the demographic group \(s\). A weighting of \(w_s(i) = 1/|A_s|\) for all \(i \in A_s\)
corresponds to average cost for group \(s\).

Previous work has shown that a bicriteria algorithm that uses \(k/(1 - \epsilon)\) number of cen-
ters can achieve a \(2^{O(p)/\epsilon}\) factor of the optimal solution [74]. In addition, [75] presents a
polynomial time algorithm that achieve \(2^{O(p)} \log^m \log \log m\) approximation and only uses \(k\) cen-
ters.

We advance on both these fronts by presenting bicriteria and constant factor approxima-
tion algorithms for the socially fair \((\ell_p, k)\)-clustering problem with \(m\) groups. In particular,
we present (1) a polynomial-time \((5 + 2\sqrt{6})^p\)-approximation with at most \(k + m\) centers (2)
a \((5 + 2\sqrt{6} + \epsilon)^p\)-approximation with \(k\) centers in time \((nk)^{2^{O(p)m^2/\epsilon}}\), and (3) a \((15 + 6\sqrt{6})^p\)
approximation with \(k\) centers in time \(k^m \cdot \text{poly}(n)\). The former is obtained by a refinement
of the iterative rounding method by solving a sequence of linear programs. The latter two
are obtained by converting a solution with up to \(k + m\) centers to one with \(k\) centers by
sparsification methods for (2) and via an exhaustive search for (3).

To compare our improvements to state-of-the-art algorithms, consider the case of \(p = 2\),
i.e., socially fair \(k\)-means. First note that for constant \(m\), if we set \(\epsilon\) small enough so that
the bicriteria result of [74] gives a solution with at most \(k + m\) centers (i.e., \(\frac{k}{1 - \epsilon} = k + m\)),
then their result can only guarantee an \(O(k)\)-approximation which is much weaker than our
constant factor approximation. In addition, the approximation guarantee of [75] degrades if the number of groups $m$ increases, while our approximation guarantees stay constant. However, the running time of our algorithms increase by increasing $m$.

We also discuss how to speed up our algorithm using a combination of coresets and $k$-means++. This allows us to run our algorithms on randomized samples of datasets while preserving the guarantees on the full-size datasets. We compare the performance of our algorithms with existing approximation algorithms on benchmark datasets, and find that our algorithms outperform existing methods.
CHAPTER 2
THE BIT COMPLEXITY OF EFFICIENT CONTINUOUS OPTIMIZATION

In this chapter, we analyze the bit complexity of efficient algorithms for fundamental optimization problems, such as linear regression, $p$-norm regression, and linear programming (LP). State-of-the-art algorithms are iterative, and in terms of the number of arithmetic operations, they match the current time complexity of multiplying two $n$-by-$n$ matrices (up to polylogarithmic factors). However, previous work has typically assumed infinite precision arithmetic, and due to complicated inverse maintenance techniques, the actual running times of these algorithms are unknown. To settle the running time and bit complexity of these algorithms, we demonstrate that a core common subroutine, known as inverse maintenance, is backward-stable. Additionally, we show that iterative approaches for solving constrained weighted regression problems can be accomplished with bounded-error preconditioners. Specifically, we prove that linear programs can be solved approximately in matrix multiplication time multiplied by polylog factors that depend on the condition number $\kappa$ of the matrix and the inner and outer radius of the LP problem. $p$-norm regression can be solved approximately in matrix multiplication time multiplied by polylog factors in $\kappa$. Lastly, linear regression can be solved approximately in input-sparsity time multiplied by polylog factors in $\kappa$. Furthermore, we present results for achieving lower than matrix multiplication time for $p$-norm regression by utilizing faster solvers for sparse linear systems.

2.1 Introduction

Over the past two decades, many breakthroughs in algorithm design have relied on continuous algorithmic primitives [76, 77, 8, 6, 7, 78]. The increased attention on continuous methods has in turn led to renewed interest and improved runtime bounds for fundamen-
tal numerical routines. Many of these bounds, however, were initially claimed assuming exact computations with infinite precision\(^1\). Relaxing this assumption, of course, is crucial for claiming truly fast(er) algorithms. As an illustration, in the case of linear systems, Krylov space methods such as the celebrated Conjugate Gradient algorithm are known to take \(O(n \cdot \text{NNZ}(A))\) arithmetic operations (where \(\text{NNZ}\) denotes the number of nonzero entries), but each involving intermediate numbers with \(\Omega(n)\) bits in the worst case. So, even though each iteration is simply a matrix-vector multiplication, the cost of each iteration is \(O(n \cdot \text{NNZ}(A))\) and the overall cost is \(O(n^2 \cdot \text{NNZ}(A))\), which makes it considerably worse than “direct methods” — linear systems can be solved with bit complexity \(\tilde{O}(n^{\omega})\) [18]. It was shown via a careful bit complexity analysis, that the block-Krylov method can be used to solve sufficiently sparse linear systems for poly-conditioned matrices faster than matrix multiplication [45, 46].

For more general convex optimization problems such as regression and linear programming, fast iterative methods have been studied intensively for decades, with much of the focus on the interior-point method (IPM) for convex optimization. Since Karmarkar [21] and Vaidya’s seminal papers [22], maintaining the inverse of a matrix modified by low-rank updates has been an important tool in fast algorithms for linear programming [6, 7, 78, 79, 80, 81], \(p\)-norm regression [77, 8, 82, 83, 84, 85], semi-definite programming [86, 87, 88] and a host of dynamic optimization problems [89, 90, 91, 92]. Recent successes in achieving asymptotic complexity close to the time required for matrix multiplication all rely on solving a slowly-changing linear system in each iteration. As a result, they lend themselves to inverse maintenance, rather than solving the linear system from scratch in each iteration.

In the optimization literature, it is widely acknowledged that the bit complexity analysis can be difficult. For example, in the paper that presents his interior-point method [93], Renegar makes the following remark about the bit complexity of the original interior point algorithm due to Karmarkar:

---

\(^1\)This is unlike the development in the classic book [5], where bit complexity analysis is carried out carefully when establishing polynomial time bounds.
“In the original version of this paper I wrote that I did not see how Kar-
markar’s algorithm could be carried out with $O(L)$ bits of accuracy (assuming
the number of bits required to represent the original problem is $L$) as Kar-
markar claimed in his paper. Subsequently, Karmarkar convinced me that this
could be done if one does not rely on rank one updates, as the algorithm in the
present paper does not. The argument, embedded in our complexity analysis,
relies on the fact that the linear equations that need to be solved need only be
solved approximately, and this can be done efficiently using Cholesky factor-
ization and the fact that the condition number of the corresponding matrices
are bounded by $2^{O(L)}$. ” [93]

Here $L$ refers to the total bit complexity of the problem (i.e., the sum of the number of
bits of all entries of $A, b, c$). Later works argued that $L$ can be replaced by the log of the
maximum subdeterminant of the constraint matrix $A$ [76, 6, 7]. Since the latter is smaller,
we adopt the latter definition for $L$ for the rest of the chapter. Let us note right away that
$L$ can be as large as $n\ell$ where $\ell$ is the number of bits used to represent any single entry of
$A$. Moreover $L = \Omega(n)$, with high probability, on random matrices [32]. This implies that
in most instances, the actual running time of these algorithms is a factor of $n$ more than
the number of arithmetic operations. In this chapter, we show that these algorithms (with
proper modifications and roundings) can be carried out with a bit complexity depending on
the logarithm of the condition number (which is $O(\log n)$, with high probability, on random
matrices [94, 95]), and the logarithm of the ratio of the outer and inner radius of the LP. In
another paper, that introduced his condition number [96], Renegar promotes the use of the
conjugate-gradient method for solving the linear systems arising in each step of the IPMs.
Note that this also leads to an extra factor of $n$.

Since matrix inverses are only computed approximately, it is important to show the
number of bits of accuracy needed to guarantee the final target accuracy remains small
despite the error accumulation during inverse maintenance. This property is closely re-
lated to what numerical analysts call stability and is carefully studied in many numerical algorithms, e.g., for the computation of eigenvalues [97]; however, it is not rigorously established for state-of-the-art results based on the IPM.

The situation is more troubling for nonlinear optimization problems such as $p$-norm minimization, i.e., $\min\{\|x\|_p : A^Tx = b\}$. For any $p \geq 2$, there are iterative algorithms that need only $\tilde{O}(n^{1/3})$ iterations (unlike LP, which currently needs $\sqrt{n}$ iterations). However, the analysis of these algorithms [8, 82, 83, 77] assumes infinite bit precision, while using sophisticated variants of inverse maintenance. Another difficulty with $p$-norm minimization is that the bit complexity of the exact solution can be unbounded (since it can be irrational), while for linear systems (and linear regression, i.e., $p = 2$), it can be bounded by $\tilde{O}(n\ell)$ using rational number representations, where $\ell$ is the bit complexity of the entries of the input matrix. It has been shown that a linear system can be solved in time $\tilde{O}(n^{\omega} \cdot \ell)$ [18]; however, the bit complexity of algorithms even for solving linear regression problems in input-sparsity time, a widely studied and important problem [19, 98, 99], is not established.

In this chapter, we address the bit complexity of state-of-the-art algorithms for linear regression, $p$-norm minimization, and linear programming. Our core technical results bound the bit complexity of general inverse maintenance and iterative algorithms for solving linear regression problems. We believe that these tools will be broadly useful for numerical algorithms in continuous optimization.

The impractical nature of matrix multiplication algorithms with exponents below 2.5 means the fastest bit complexity bounds we state are only of theoretical interest. However, our results apply to all matrix multiplication algorithms up to the current fastest one, with matrix exponent $\omega \approx 2.372$ [16]. In particular, they apply to algorithms with $O(n^3)$ and $O(n^{2.808})$ (Strassen’s algorithm [14]) running times. Moreover, the iterative algorithms we consider are remarkably effective in practice. For example, our analyses in Section 2.4 is based on the interior point method used in the Gurobi library \(^2\); iterative refinement

approaches for \( p \)-norm regression have shown promising results in practice [82]; and iterative approaches for linear regression have resulted in theoretically and practically faster algorithms for tensor decomposition [57].

2.1.1 Results

We start with some definitions related to the stability and bit complexity of algorithms. The condition number of a function \( f : \mathbb{R}^n \rightarrow \mathbb{R}^m \) is defined as the smallest nonnegative real number \( \kappa_f \) such that

\[
\frac{\|f(x + \delta x) - f(x)\|}{\|f(x)\|} = \kappa_f \cdot \frac{\|\delta x\|}{\|x\|} + O\left(\left(\frac{\|\delta x\|}{\|x\|}\right)^2\right),
\]

where \( x, \delta x \in \mathbb{R}^n \). For the inverse of matrices, this coincides with the condition number of the matrix defined as \( \kappa(A) := \|A\|_2 \cdot \|A^{-1}\|_2 \). It has been shown that a recursive algorithm based on fast matrix multiplication is logarithmically stable [17] in the following sense.

\[
\frac{\|g(A) - A^{-1}\|}{\|A^{-1}\|} \leq O(\epsilon) \kappa(A)^{\text{poly log}(n)} + O(\epsilon^2),
\]

where \( g(A) \in \mathbb{R}^{n \times n} \) is the output of the algorithm for the inverse. Taking

\[
\epsilon < \epsilon \|A^{-1}\| \kappa(A)^{\text{poly log}(n)},
\]

we can guarantee that \( \|g(A) - A^{-1}\| \leq \epsilon \). This implies that by using \( \text{poly log}(n) \cdot \log(\kappa) + \log(1/\epsilon) \) bits, we can achieve \( \|g(A) - A^{-1}\| \leq \epsilon \). Therefore we consider the following definitions of numerical stability that are equivalent up to a \( \kappa(A) \) factor.

**Definition 2.1.1.1** (Numerical Stability of Computing the Inverse). Let \( A \in \mathbb{R}^{n \times n} \) be an invertible matrix. Consider an algorithm that computes the matrix \( M \in \mathbb{R}^{n \times n} \) as the inverse of \( A \). Then the algorithm is forward stable if \( \|M - A^{-1}\| \leq \epsilon \), and it is backward stable if \( \|M^{-1} - A\| \leq \epsilon \).
As we will discuss later, forward stability suffices for solving linear regression. However, for inverse maintenance guarantees, we require backward stability.

Our first result bounds the bit complexity of solving a linear regression problem in input-sparsity time. We later extend this to certain weighted constrained regression problems that are used as a subprocedure for the $p$-norm regression problem. We will use running time for the total time counting bit-level operations and bit complexity to refer to the size of representations. We also use “with high probability” to mean with probability at least $1 - n^{-C}$ for any constant $C$.

**Theorem 2.1.1.2 (Linear Regression).** Let $A \in \mathbb{R}^{n \times d}$ be a full-rank matrix with $n \geq d$ and a condition number bounded by $\kappa$, and $b \in \mathbb{R}^n$ all with bit complexity of $\log(\kappa)$. Let $x^* = \arg\min_x \|Ax - b\|_2 = (A^T A)^{-1} A^T b$ and $0 < \epsilon < 1$. Then there is an algorithm that, with high probability, computes $\hat{x}$ such that

$$\|\hat{x} - x^*\|_{A^T A} \leq \epsilon \cdot \|A(A^T A)^{-1} A^T b\|_2,$$

or equivalently

$$\|A\hat{x} - b\|_2 \leq \epsilon \cdot \|A(A^T A)^{-1} A^T b\|_2 + \|(I - A(A^T A)^{-1} A^T) b\|_2,$$

in time $\tilde{O}((d^w + d^2 \cdot \log^2(1/\epsilon)) + \text{NNZ}(A) \cdot \log^2(1/\epsilon)) \cdot \log \kappa)$.

The $\tilde{O}$ in the above result and the rest of the chapter hides poly log$(nd)$ factors and poly log log$(\kappa/\epsilon)$ factors. For simplicity, in all of our results, we assume the matrix $A$ has full column rank. However, this is not a limitation. As we show in Section 2.6, low-rank matrices can be modified to matrices with full column rank by concatenating a small factor of the identity matrix that introduces an small error.

We next consider the bit complexity of approximately solving linear programs. We
consider LPs of the following form:

\[
\begin{align*}
\min \quad & c^\top x \quad \text{(primal)} \quad \text{and} \quad \max \quad & b^\top y \quad \text{(dual)}.
\end{align*}
\]

We start by defining a few parameters.

**Definition 2.1.1.3.** Let \( A \in \mathbb{R}^{n \times d}, b \in \mathbb{R}^d, c \in \mathbb{R}^n \) with \( n \geq d \). For a linear program of the form \( \min_{A^\top x = b, x \geq 0} c^\top x \), we define the following quantities:

- **Inner radius** \( r \): There exists an \( x \) such that \( A^\top x = b \) and \( x_i \geq r \geq 0 \) for all \( i \in [n] \).
- **Outer radius** \( R \): For all \( x \geq 0 \) with \( A^\top x = b \), \( \|x\|_2 \leq R \).

The next theorem states that the robust IPM [6, 7] only requires numbers with \( \tilde{O}(\log(\frac{R}{\epsilon R})) \) bits in fixed-point arithmetic. We note that directly utilizing algorithms of [6, 7], does not imply the time complexity of the following result. First, the bit complexity of inverse maintenance has to be bounded (with proper rounding at update steps — see the data structure in Algorithms 4 and 5) and second, the modifications made to the problem to find an initial feasible solution, should be made in a way that ensures the condition number of the constraint matrix does not change significantly. We adopt the initialization approach of [100] and show that the condition number of the resulting matrix stays the same up to polynomial factors in \( n \).

**Theorem 2.1.1.4 (Robust IPM).** Given \( A \in \mathbb{R}^{n \times d} \) with full column-rank and condition number \( \kappa \), \( b \in \mathbb{R}^d \), \( c \in \mathbb{R}^n \) all with bit complexity of \( \log(\kappa) \), and an error parameter \( 0 < \epsilon < 1 \), suppose the inner radius and outer radius of the linear program \( \min_{A^\top x = b, x \geq 0} c^\top x \) is \( r \) and \( R \), respectively. Then there is an algorithm that computes \( \hat{x} \in \mathbb{R}^n \) such that

\[
\begin{align*}
&c^\top \hat{x} \leq \min_{A^\top x = b, x \geq 0} c^\top x + \epsilon, \text{ and } \|A^\top \hat{x} - b\|_2 \leq \epsilon,
\end{align*}
\]

in time \( \tilde{O} \left( \left( n^\omega + n^{2.5-\alpha/2} + n^{2+1/6} \right) \cdot \log \left( \frac{R}{\epsilon R} \right) \cdot \log \left( \frac{R}{\epsilon R} \right) \right) \).
We only assume that the bit complexity of $A, b, c$ is bounded by $\log(\kappa)$ for ease of notation. If the bit complexity of them is $\ell$, the first log factor will be replaced by $\ell + \log\left(\frac{R}{e^r}\right)$. Note that our bit complexity depends on $\log(\kappa)$ as opposed to the bit complexity stated in [6], which is the logarithm of the maximum determinant over the square submatrices. Note that although both quantities are $\Omega(n)$ in the worst case, for random matrices, the latter is $\Omega(n)$ while the former is $O(\log n)$. This is because, for random matrices, the condition number is polynomially bounded [94, 95] while the determinant is exponentially large [32] with high probability. Moreover, $\log(R/r)$ has shown to be $O(\log n)$ in the smoothed analysis of LPs [101]. Finally, note that we are concerned with approximate solutions to LPs. An exact solution might require the bit complexity proportional to the logarithm of the maximum determinant of square submatrices. The exponent of the third term above is recently improved to $2 + \frac{1}{18}$ by using more complicated data structures [80].

The above approach is not always the fastest algorithm for solving LPs approximately. The next result is based on solving linear systems using shifted number systems [18], which avoids the $\log(\kappa)$ factor. This approach does not use inverse maintenance techniques.

**Theorem 2.1.1.5** (Inverse-free IPM). Given $A \in \mathbb{R}^{n \times d}$ with full column-rank, $b \in \mathbb{R}^d$, $c \in \mathbb{R}^n$ all with bit complexity of $\ell$, and an error parameter $0 < \epsilon < 1$, suppose the inner radius and outer radius of the linear program $\min_{\mathbf{A}^\top \mathbf{x} = \mathbf{b}, \mathbf{x} \geq 0} \mathbf{c}^\top \mathbf{x}$ is $r$ and $R$, respectively. Then there is an algorithm that finds $\hat{x} \in \mathbb{R}^n$ such that

$$\mathbf{c}^\top \hat{x} \leq \min_{\mathbf{A}^\top \mathbf{x} = \mathbf{b}, \mathbf{x} \geq 0} \mathbf{c}^\top \mathbf{x} + \epsilon, \quad \|\mathbf{A}^\top \hat{x} - \mathbf{b}\|_2 \leq \epsilon,$$

in time $\tilde{O}(n^{\omega+0.5} \cdot (\ell + \log(\frac{R}{e^r})) \cdot \log(\frac{R}{e^r}))$.

The algorithm of Theorem 2.1.1.5 is faster than Theorem 2.1.1.4 by a factor of $n^{0.5}$ when $\ell = O(1)$, $\log(\kappa) = \Omega(n)$, and $\log(R/r) = O(\log n)$. We discuss such a case in Section 2.4.1. This highlights the fact that when we consider the actual running time of algorithms, algorithms with smaller number of arithmetic operations do not necessarily
have the smallest running time. Our final result for LPs is presented in Theorem 2.4.0.1 and shows one can go below matrix multiplication time for \( \omega > 2.5 \) and sparse poly-conditioned matrices.

We next turn to \( p \)-norm minimization problems for \( p \geq 2 \). All of our results can be extended to the case of \( p \in (1, 2] \) by considering the dual norm using the approach explained in Section 7 of [8]. Our first result bounds the bit complexity of solving the \( p \)-norm problem in both sparse and dense cases. Since the only difference between the two cases is the data structure we use, we present both of them in a single theorem.

**Theorem 2.1.1.6** (\( p \)-norm minimization). Let \( A \in \mathbb{R}^{n \times d} \) be a matrix with condition number bounded by \( \kappa \), and \( b \in \mathbb{R}^d \) be a vector with the bit complexity bounded by \( \log(\kappa) \). Let \( x^* = \arg \min_{A^\top x = b} \|x\|^p_p \). For \( p \geq 2 \), there is an algorithm that computes \( \hat{x} \) such that

\[
\|\pi_A (\hat{x} - x^*)\|_2 \leq \epsilon \|\pi_A x^*\|_2,
\]

and

\[
\|\hat{x}\|_p^p \leq (1 + \epsilon) \|x^*\|_p^p
\]

in time

\[
\tilde{O}_p \left( (n^\omega + n^{7/3} \cdot \log(1/\epsilon)) \log^{1.5} (\kappa/\epsilon) \log^2 (1/\epsilon) \right).
\]

Moreover, for sparse matrices, there is an algorithm that returns an output with the same guarantees, with high probability, in time

\[
\tilde{O}_p \left( n^{7/3} \cdot \left( 1 + \text{NNZ}(A) \frac{\omega^{-7/3}}{\omega - 1} \right) \log^{2.5} (\kappa/\epsilon) \log^3 (1/\epsilon) \right).
\]

The subscript \( p \) hides a function \( f(p) \). For any value of \( \omega > 7/3 \), \( \text{NNZ}(A) = o(n^{\omega - 1}) \), and \( \log(\kappa/\epsilon) = \text{poly}(n) \), the above gives a running time \( o_p(n^\omega) \). For example, for polyconditioned matrices with \( \text{NNZ}(A) = O(n) \) and the current value of \( \omega \approx 2.372 \), the running time is \( \tilde{O}_p(n^{2.363} \cdot \log^{5.5} (\epsilon^{-1}) \) \). Note that the powers of \( \log(1/\epsilon) \) and \( \log(\kappa/\epsilon) \) are different for \( p \)-norm and linear programming. This is because of the number of iterations of
the algorithms arising from “guessing” the optimal values in subprocedures of our \( p \)-norm regression algorithm. Moreover note that for \( p \)-norm problem, we also modify the matrices by concatenating a (gradient) vector. We prove that this only affects the condition number of the matrix by a polynomial factor — see Section 2.5.2. We also emphasize that we use a different approach than [8] for solving the constrained weighted regression problems, that are subprocedures of the algorithm, to be sure that the numbers we work with only have \( \log(\kappa/\epsilon) \) bits — see Section 2.5.3. Note that taking the powers of \( p \) of the numbers in the algorithm only increases the bit complexity by a factor of \( p \) which is absorbed in the \( \tilde{O}_p \) notation.

Our approach for solving the \( p \)-norm minimization problems is to solve a series of smoothed \( p \)-norm minimization problems (see Sections 2.5.1 and 2.5.2) to constant factor approximation. The smoothed \( p \)-norm problem, which we also refer to as mixed \((2, p)\)-norm minimization, is defined as follows.

**Definition 2.1.1.7** (Smoothed \( p \)-norm minimization problem). Let \( A \in \mathbb{R}^{n \times d} \) and \( b \in \mathbb{R}^d \), with \( n \geq d \). For \( p \geq 2 \), let

\[
x^* = \arg \min_{x \in \mathbb{R}^n : Ax = b} \sum_{i=1}^n \gamma_p(t_i, x_i),
\]

where for \( t \in \mathbb{R}_{\geq 0} \) and \( x \in \mathbb{R} \),

\[
\gamma_p(t, x) := \begin{cases} 
\frac{p}{2} t^{p-2} x^2 & \text{if } |x| \leq t, \\
|x|^p + \left( \frac{p}{2} - 1 \right) t^p & \text{otherwise}.
\end{cases}
\]

Then the smoothed \( p \)-norm problem asks for \( \hat{x} \) such that \( \|\pi_A (\hat{x} - x^*)\|_2 \leq \epsilon \cdot \|\pi_A x^*\|_2 \) and

\[
\sum_{i=1}^n \gamma_p(t_i, \hat{x}_i) \leq (1 + \epsilon) \cdot \sum_{i=1}^n \gamma_p(t_i, \tilde{x}_i).
\]

We show that the following mixed \((2, \infty)\)-norm minimization problem can be used as a
proxy for such smoothed $p$-norm problems, but this leads to larger running times for solving the $p$-norm minimization problem. However, since mixed $(2, \infty)$-norm minimization is an important problem in its own right, we present a multiplicative weights update algorithm for it as well.

**Definition 2.1.1.8** (Mixed $(2, \infty)$-norm minimization problem). Let $A \in \mathbb{R}^{n \times d}$, $b \in \mathbb{R}^d$, $r, s \in \mathbb{R}^n_{\geq 0}$ with $n \geq d$. Let

$$x^* = \arg \min_{x \in \mathbb{R}^n : A^\top x = b} \|x\|_r^2 + \|s \odot x\|_{\infty},$$

where $\odot$ is the entrywise (Hadamard) product, $\|x\|_r^2 = x^\top R x$, and $R$ is the diagonal matrix corresponding to $r$. Then the mixed $(2, \infty)$-norm minimization problem asks for $\tilde{x}$ such that

$$\left\| \pi_A (\tilde{x} - x^*) \right\|_2 \leq \varepsilon \cdot \|\pi_A x^*\|_2$$

and

$$\left\| \tilde{x} \right\|_r^2 + \|s \odot \tilde{x}\|_{\infty} \leq (1 + \epsilon) \cdot \left( \left\| x^* \right\|_r^2 + \|s \odot x^*\|_{\infty} \right).$$

We provide a constant factor approximation algorithm for this problem.

**Theorem 2.1.1.9** (Mixed $(2, \infty)$-norm minimization). Let $A \in \mathbb{R}^{n \times d}$ and $b \in \mathbb{R}^d$, $r, s \in \mathbb{R}^n_{\geq 0}$, $n \geq d$, such that the condition number of $A$ is less than $\kappa$ and the bit complexity of all of them is bounded by $\log(\kappa)$. For $0 < \varepsilon < 1$, there is an algorithm that outputs $\tilde{x}$ such that

$$\left\| \pi_A (\tilde{x} - x^*) \right\|_2 \leq \varepsilon \cdot \|\pi_A x^*\|_2$$

and

$$\left\| \tilde{x} \right\|_r^2 + \|s \odot \tilde{x}\|_{\infty} = O(1) \cdot (\| x^* \|_r^2 + \|s \odot x^*\|_{\infty}),$$

where $x^* = \arg \min_{x : A^\top x = b} \|x\|_r^2 + \|s \odot x\|_{\infty}$, in time

$$O_p ((n^{\alpha_2} + n^{7/3} \cdot \log^2 (1/\varepsilon)) \log (\alpha_2 \kappa / \varepsilon) \log (\alpha_1 \kappa) \log (\kappa / \varepsilon),$$

where $\alpha_1 = 1/(\min_{i \in [n]} r_i + s_i^2)$ and $\alpha_2 = (\max_{i \in [n]} r_i + \max_{i \in [n]} s_i) / \min_{i \in [n]} r_i$. Moreover,
for sparse matrices, there is an algorithm that returns an output with the same guarantees with probability at least $1 - n^{-10} \cdot \log(\alpha_2)$ in time

$$
\tilde{O}
\left(
\left(n^{7/3} \cdot \left(1 + \text{NNZ}(A)^{-\omega^{-1}/\omega+1}\right)\log^2(1/\epsilon) \log(\alpha_2/\epsilon) \log(\alpha_1/\kappa) \log^2(\kappa/\epsilon)\right)
\right).
$$

We finally note that, while the bit complexity of these problems is known for Laplacians and graph problems such as maximum flow, it was not known for general matrices prior to our work. The main reasons for this difference are the use of inverse maintenance techniques for general matrices and the difficulty of establishing bounds on the condition number of such matrices.

### 2.1.2 Techniques

Inverse maintenance is an important technique that has been used in optimization algorithms since Karmarkar [21]. It has since been utilized in many other algorithms, such as iterative refinement for $p$-norm minimization and dynamic algorithms. The following identity, which is used for inverse maintenance, has been extensively used (without stability and bit complexity analysis) in optimization literature to speed up a variety of different iterative algorithms.

**Fact 2.1.2.1** (Sherman-Morrison-Woodbury identity [102]). For an invertible $n \times n$ matrix $M$ and matrices $U \in \mathbb{R}^{n \times r}$, $C \in \mathbb{R}^{r \times r}$, $V \in \mathbb{R}^{r \times n}$, we have

$$(M + UCV)^{-1} = M^{-1} - M^{-1}U(C^{-1} + VM^{-1}U)^{-1}VM^{-1}.$$
Our first main technique is to show that inverse maintenance via the Sherman-Morrison-Woodbury identity is backward stable. We need to present our numbers with \( \tilde{O}(\log(\kappa/\epsilon)) \) bits to have this guarantee. The following lemma states that after applying the Woodbury identity, the backward error only increases additively in each iteration. Therefore, if we apply this method for \( \text{poly}(n) \) iterations, the error only increase by \( \text{poly}(n, \kappa) \cdot \epsilon \). Therefore by picking \( \epsilon \) to be appropriately small, we can guarantee that the inverse has small error over the course of an algorithm with \( \text{poly}(n) \) iterations, such as interior point methods [6, 7] and multiplicative weights update methods [8].

**Lemma 2.1.2.2** (Backward Stability of Inverse Maintenance). Let \( Z \in \mathbb{R}^{n \times n}, \bar{Z} \in \mathbb{R}^{n \times n}, C \in \mathbb{R}^{m \times m} \) be invertible matrices. Moreover let \( U, V \in \mathbb{R}^{n \times m} \) such that \( Z + UCV^T \) is invertible. Let \( \kappa > m + n \) such that

\[
\|U\|_F, \|V\|_F, \|C\|_F, \|C^{-1}\|_F, \|Z\|_F, \|Z^{-1}\|_F, \|(Z + UCV^T)^{-1}\|_F \leq \kappa
\]

and \( 0 < \epsilon_1, \epsilon_2 < 1 \). Suppose

\[
\|\bar{Z} - Z\|_F \leq \epsilon_1. \tag{2.1.2}
\]

If \( D \in \mathbb{R}^{m \times m} \) is an invertible matrix such that

\[
\left\|D^{-1} - (C^{-1} + V^T\bar{Z}^{-1}U)^{-1}\right\|_F \leq \epsilon_2,
\]

then

\[
\left\|(\bar{Z}^{-1} - \bar{Z}^{-1}UD^{-1}V^T\bar{Z}^{-1})^{-1} - (Z + UCV^T)^{-1}\right\|_F \leq 512\kappa^{26}\epsilon_2 + \epsilon_1.
\]

In addition to inverse maintenance, for the \( p \)-norm minimization problem we need high-accuracy solutions given a constant factor spectral approximation as the preconditioner. We note that even in the cases where we only solve one static linear regression problem (as opposed to a series of dynamically changing linear regression problems like algorithms for...
$p$-norm regression), one might need to use an iterative approach based on preconditioning instead of a direct solve to obtain a high-accuracy solution in certain running times. An example of this is high-accuracy input-sparsity time algorithms for solving linear regression problems [19, 99]. Another example is illustrated by [57] in the context of tensor decompositions in which the algorithm requires a preconditioning approach to achieve a speed-up to subquadratic time.

**Lemma 2.1.2.3** (High-accuracy solutions for constrained weighted linear regression). Let $A \in \mathbb{R}^{n \times d}$ have full column rank, $b \in \mathbb{R}^d$, and $W \in \mathbb{R}^{n \times n}$ be a diagonal matrix with $RI \succeq W \succeq I$. Moreover let $x^* = \arg\min_{x : A^\top x = b} \frac{1}{2} \|x\|^2_W$. Then

$$x^* = W^{-1} A(A^\top W^{-1} A)^{-1} b.$$

Moreover given a matrix $\tilde{M}^{-1}$ such that there exists matrix $M$ with $\|\tilde{M}^{-1} - M^{-1}\|_F \leq \frac{\epsilon}{d \lambda \|A^\top W^{-1} A\|_2}$ and $A^\top W^{-1} A \preceq M \preceq \lambda A^\top W^{-1} A$ with a constant $\lambda \geq 1$, there is an algorithm that finds $\tilde{x}$ such that

$$\|\tilde{x} - x^*\|_2 \leq \epsilon \cdot \|x^*\|_2, \quad \|\tilde{x}\|_W \leq (1 + \epsilon) \|x^*\|_W, \quad \text{and} \quad \|\pi_A(\tilde{x} - x^*)\|_2 \leq \epsilon \|\pi_A x^*\|_2,$$

where $\pi_A$ is the projection matrix of matrix $A$, in $O((d^2 + \text{NNZ}(A)) \cdot \log(\kappa(A) \cdot R) \cdot \log^2(\frac{\kappa}{\epsilon}))$ time.

Note that since $\|M^{-1}\| \leq \text{poly}(nRk/R)$, we can take bit complexity of $O(\log(nRk/\epsilon))$ for the matrix $\tilde{M}^{-1}$ to satisfy the condition $\|\tilde{M}^{-1} - M^{-1}\|_F \leq \frac{\epsilon}{d \lambda \|A^\top W^{-1} A\|_2}$. A complication in Lemma 2.1.2.3 is that we require a vector $\tilde{x}$ that is close to $x^*$ in two different norms: one induced by $W$ and the other induced by $\pi_A$. Interestingly, as we show, one does not need to take $\log(\kappa)$ iterations to achieve this.
2.1.3 Discussion

Although the running times of optimization algorithms in terms of number of arithmetic operations have been extensively studied in the past decades, in many recent works, the bit complexity is left unanalyzed. [6] and [8] present algorithms solving linear programs and $p$-norm minimization problems respectively with running times that match the matrix multiplication time $n^{\omega}$ up to polylogarithmic factors. However even solving one linear system under fixed-point arithmetic, by computing the inverse and applying it to the vector, requires bit complexity of $\Omega(\log(\kappa/\epsilon))$ even if the bit complexity of the original linear system is $O(1)$. This is exemplified by the following matrix that has a condition number of larger than $2^{n-1}$, by testing the vectors

$$
\begin{bmatrix}
1 & 0 & 0 & \cdots & 0
\end{bmatrix}, \quad \text{and} \quad \begin{bmatrix}
(-1/2)^{n-1} & (-1/2)^{n-2} & \cdots & -1/2 & 1
\end{bmatrix}
$$

for the largest and smallest singular value, respectively.

$$
\begin{bmatrix}
1 & 0 & 0 & \cdots & 0 & 0 \\
2 & 1 & 0 & \cdots & 0 & 0 \\
0 & 2 & 1 & \cdots & 0 & 0 \\
0 & 0 & 2 & 1 & \cdots & 0 \\
\vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\
0 & 0 & 0 & 0 & \cdots & 1 \\
0 & 0 & 0 & 0 & \cdots & 2 \\
1 &
\end{bmatrix} \in \mathbb{R}^{d \times d}.
$$

It is not a priori clear what bit complexity is required to guarantee convergence when we need to solve a series of dynamically changing linear systems as required by iterative approaches for solving $p$-norm minimization and LPs. Although the forward stability of the inverse maintenance processes has been considered [103], such bounds are not enough for algorithms that need poly$(n)$ iterations.
Note that as illustrated by the above example, an algorithm with $O(n^\omega)$ arithmetic operations and bit complexity of $\log(\kappa)$, in the worst case, has a running time of $O(n^{\omega+1})$. Therefore it is crucial to determine the right values for the power of $\log(\kappa)$ factor. Additionally as illustrated by Theorem 2.1.1.5, an algorithm with smaller number of arithmetic operations does not necessarily have the best overall running time.

**Outline.** We start by presenting our result on input-sparsity time linear regression in Section 2.2. We then present our numerically stable inverse maintenance in Section 2.3. Our data structures that use this inverse maintenance procedure for dense and sparse matrices are presented in Section 2.3.1. Equipped with these, we present our results on solving linear programs in Section 2.4. We first discuss our overall algorithm and how to find the initial feasible solution in Section 2.4.1. We then present our LP solvers that uses robust IPM in Section 2.4.2, our LP solver based on shifted numbers in Section 2.4.3, and our results for sparse LP and $\omega > 2.5$ in Section 2.4.4.

We introduce the outer loop of our algorithm for solving the $p$-norm minimization problem that uses a series of solutions to residual problems in Section 2.5.1. Then in Section 2.5.2, we discuss how the residual problem can be solved effectively by solving instances of smoothed $p$-norm minimization problems and how the mixed $(2, \infty)$-norm minimization can be used as a proxy. We then present our multiplicative weights update (MWU) algorithm to solve a mixed $(2, \infty)$-norm minimization problem in Section 2.5.4. Finally, in Section 2.5.5, we present our MWU algorithm for solving the smoothed $p$-norm minimization problem.

**2.1.4 Notation and Preliminaries**

**Linear algebra notations.** We denote the Hadamard (entrywise product) with $\odot$. For a vector $x$, let $|x|$ be a vector of same size such that $(|x|)_i = |x_i|$ for all $i$; and $x^p$ denotes the vector with its $i$th entry equal to the $i$th entry of $x$ to the power of $p$, i.e., $(x^p)_i = (x_i)^p$. 

37
Similarly, for a diagonal matrix (or vector) $M$, $\sqrt{M}$ is a matrix where each entry is equal to the square root of the corresponding entry in $M$. For a matrix $A$ with $n$ rows and a subset $S \subseteq [n]$, let $A_S$ denote the matrix obtained by taking rows of $A$ with indices in $S$. $A_{S}$ denote the matrix obtained by taking the columns of $A$ with indices in $S$. For a square matrix $A \in \mathbb{R}^{n \times n}$ and $S \subseteq [n]$, $A_{S,S}$ denotes the matrix obtained by taking entries of $A$ in $S \times S$. Note that, we apply these subindices before taking transpose, i.e., $A_S^T = (A_S)^T$. We denote the Moore-Penrose inverse (i.e., pseudoinverse) of $A$ with $A^\dagger$.

$\|\cdot\|_F$ denotes the Frobenius norm. We denote the entrywise norm of matrices by $\|\cdot\|$, e.g., $\|A\|_\infty$ is the maximum magnitude over entries of $A$. For a matrix $A$, we denote its condition number by $\kappa(A) := \|A\|_2 \|A^\dagger\|_2$. In other words, the condition number of a matrix is its largest singular value divided by its smallest nonzero singular value. We denote the orthogonal projection matrix of $A$ with $\pi_A$. In particular, if $A$ has full column rank, $\pi_A = A(A^\top A)^{-1}A^\top$. Throughout the chapter, to make the notation less cumbersome, we assume the bit complexity of the vector $b$ and matrix $A$ are at most $\log(\kappa)$. This means that the absolute value of each entry of $b$ and $A$ is either zero or in the interval $[\frac{1}{\kappa}, \kappa]$. This is without loss of generality since the factors of the running time depending on $\kappa$ can be replaced with $\log((\kappa \cdot 2^\ell)/\epsilon)$, where $\ell$ is the bit complexity of the input.

When it is clear from the context, we denote the diagonal matrix corresponding to a vector with the capital letter of the vector, e.g., $W$ denotes the diagonal matrix corresponding to $w$. Also for $u, w \in \mathbb{R}^n$, we define

$$\|u\|_w = \|u\|_W := \sqrt{u^\top W u},$$

for $w \geq 0$. More generally for a symmetric positive semi-definite matrix $M$ we denote $\|u\|_M = \sqrt{u^\top M u}$. For vectors $u_1 \in \mathbb{R}^{n_1}, \ldots, u_k \in \mathbb{R}^{n_k}$, we denote by $(u_1, \ldots, u_k) \in \mathbb{R}^{n_1 + \cdots + n_k}$, their concatenation. Note that $n_i$ could be equal to one, in which case $u_i$ is a number. For a number $t \in \mathbb{R}$, we denote the vector with all entries equal to $t$ with \[ ... \]
The dimension of the vector will be clear from the context, e.g., if $\mathbf{A} \in \mathbb{R}^{n \times d}$, $[\mathbf{A} \ 0]$ denotes a matrix obtained by attaching a column of all zeros to the matrix $\mathbf{A}$. For a function $f : \mathbb{R} \rightarrow \mathbb{R}$ and vector $\mathbf{u} \in \mathbb{R}^{n}$, we define $(f(\mathbf{u}))[i] = f(\mathbf{u}_i)$, i.e., we extend $f$ to $f : \mathbb{R}^{n} \rightarrow \mathbb{R}^{n}$.

**Matrix multiplication.** We denote the matrix multiplication exponent and its dual with $\omega$ and $\alpha$, respectively. Moreover, we denote the cost of multiplying an $n$-by-$m$ matrix with an $m$-by-$d$ matrix with MM$(n, m, d)$, e.g., MM$(n, n, n) = n^\omega$, and MM$(n, n, n^\alpha) = n^{2+o(1)}$. We need the following lemma to bound the running time of rectangular matrix multiplication (for inverse maintenance) in our running time.

**Lemma 2.1.4.1** ([104, 6]). Let $n \geq d$. Then multiplication of an $n \times d$ matrix with a $d \times n$ matrix or an $n \times n$ matrix with an $n \times d$ matrix can be performed in the following running time.

$$n^{2+o(1)} + d^{(\omega-2)/(1-\alpha)}n^{2-\alpha:(\omega-2)/(1-\alpha)+o(1)}.$$ 

For ease of notation, we drop $o(1)$ in the running time of matrix multiplication throughout the chapter.

**General assumptions.** We now state a few preliminary results to establish our assumptions in this chapter.

**Remark 2.1.4.2.** Let

$$\mathbf{x}^* = \arg\min_{\mathbf{x} : \mathbf{A}^\top \mathbf{x} = \mathbf{b}} \|\mathbf{x}\|_p^p.$$ 

For $p \geq 2$, if $\|\mathbf{x}^*\|_p \leq \varepsilon$, then $\|\mathbf{x}^*\|_2^p \leq d^{(p-2)/2}\varepsilon$. Therefore without loss of generality, we can assume $\|\mathbf{x}^*\|_p^p > 1/{\text{poly}(\kappa)}$, since otherwise $\mathbf{x} = 0$ will have a small error both in terms of the $p$-norm objective and in terms of $\|\mathbf{A}^\top \mathbf{x} - \mathbf{b}\|_2$.

The next lemma (proven in the Section 2.6) states that we only need to focus on full column rank matrices.
Lemma 2.1.4.3. Let \( p \geq 2, A \in \mathbb{R}^{n \times d}, b \in \mathbb{R}^{d}, n \geq d, \) such that the smallest nonzero singular value of \( A \) is equal to \( \sigma > 0. \) Moreover let \( 0 < \varepsilon_1 < 1 \) and \( \varepsilon_2 = \varepsilon_1 \cdot \frac{\sigma}{2 \cdot d \cdot (p-2)/2p}. \) Let

\[
\overline{A} = \begin{bmatrix} A \\ \varepsilon_2 I \end{bmatrix}.
\]

Moreover let \( 0 < \varepsilon_3 < 1, x^* \in \mathbb{R}^n \) and \( \tilde{x} \in \mathbb{R}^{n+d} \) such that

\[
x^* = \arg \min_{x: A^T x = b} \|x\|_p^p, \quad \|\tilde{x}\|_p^p \leq (1 + \varepsilon_3) \min_{x: A^T x = b} \|x\|_p^p, \quad \text{and} \quad \left\| A^T \tilde{x} - b \right\|_2 \leq \varepsilon_3.
\]

Let \( \tilde{x} \in \mathbb{R}^d \) be a vector with entries equal to the first \( n \) entries of \( \tilde{x}. \) Then

\[
\left\| A^T \tilde{x} - b \right\|_2 \leq \varepsilon_3 + \varepsilon_1 \cdot \|b\|_2, \quad \text{and} \quad \|\tilde{x}\|_p^p \leq (1 + \varepsilon_3) \|x^*\|_p^p.
\]

Remark 2.1.4.4. \( \overline{A} \) in Lemma 2.1.4.3 has full column rank. Moreover, to achieve an error of \( \varepsilon, \) we can pick \( \varepsilon_3 = \varepsilon/2 \) and \( \varepsilon_1 = \varepsilon/(2 \|b\|_2). \) Therefore \( \kappa(\overline{A}) \leq 4 \cdot d^{(p-2)/2p} \kappa(A) \|b\|_2. \) Thus solving the problem with \( \overline{A} \) only needs a polylogarithmic factor increase in bit complexity. Also since \( n + d \leq 2n, \) the polynomial factors in \( n \) of running time only increase by constant factors. Therefore for the rest of the chapter, without loss of generality, we assume the matrix \( A \) has full column rank.

Remark 2.1.4.5. Since \( A \) has full column rank if \( n = d, \) then \( A^T x = b \) has a unique solution \( x^*, \) and we can compute a vector \( \tilde{x} \) (by solving \( A^T x = b \)) that is close to \( x^* \) and with appropriate accuracy and bit complexity, we can guarantee both \( \|A^T \tilde{x} - b\| \leq \epsilon \) and \( \|\tilde{x}\|_p^p \leq (1 + \epsilon) \|x^*\|_p^p. \) Therefore for the rest of the chapter, we assume \( n > d. \)

Inverse maintenance. The following directly follows from Fact 2.1.2.1 and is one of the main tools for the robust IPM [6, 7] to obtain a solution with about \( n^\omega \) arithmetic operations.

Corollary 2.1.4.6. Let \( P = A(A^T (V + Q)A)^{-1} A^T \) where \( V \) is a diagonal matrix, and \( Q \) be
a sparse diagonal matrix with $T = \text{supp}(Q)$. Then,

$$A(A^T(V + Q)A)^{-1}A^T = P - P_T(Q_{T,T}^{-1} + P_{T,T})^{-1}P_{T,T}^T.$$  

Proof. We have $A^TQA = A_T^TQ_TTA_T$ because the only nonzero entries of $Q$ are the ones with indices in $T$. By Fact 2.1.2.1, we have

$$A(A^TVA + A^TQA)^{-1}A^T = A(A^TVA + A_T^TQ_TTA_T)^{-1}A^T = A \left( (A^TVA)^{-1} - (A^TVA)^{-1}A_T^T(Q_{T,T}^{-1} + A_T(A^TVA)^{-1}A_T)^{-1}A_T(A^TVA)^{-1} \right)A^T.$$  

The result follows by observing that $A_T(A^TVA)^{-1}A_T^T = P_{T,T}$ and $A(A^TVA)^{-1}A_T^T = P_{T,T}$. \hfill \Box

2.2 Linear Regression

For a linear regression problem $\min_x \|Ax - b\|_2$ with $x^* = \arg \min_x \|Ax - b\|_2$, we might want to make $\|x - x^*\|_2$ or $\|A(x - x^*)\|_2 = \|x - x^*\|_{A^TA}$ small. These are different for several reasons. For example, $x^*$ might not be unique (if $A$ is not full-rank), but $Ax^*$ is unique. Even in the case where $x^*$ is unique $\|x - x^*\|$ might be large while $\|x - x^*\|_{A^TA}$ is small, e.g., when $x - x^*$ is in the direction of the right singular vector of $A$ corresponding to the smallest singular value of $A$. However note that $\sigma_{\min}^2 I \leq A^TA \leq \sigma_{\max}^2 I$, where $\sigma_{\min}$ and $\sigma_{\max}$ are the smallest and largest singular values of $A$. Therefore assuming $\sigma_{\min} \leq 1 \leq \sigma_{\max}$ (which can be achieved by scaling), $\|x - x^*\|_2$ is within a $\kappa(A)^2$ factor of $\|x - x^*\|_{A^TA}$.

In many applications, the goal is to bound $\|x - x^*\|_{A^TA}$ directly. For example, see Corollary 2.2.0.5 which is a vector that is computed in each iteration of interior-point methods for solving linear programs. This then can be achieved by making sure $\|x - x^*\|_2 \leq \frac{\epsilon}{\kappa(A)}$ or directly bounding $\|x - x^*\|_{A^TA}$. Even irrespective of bit complexity, the former
might need $\log(\kappa/\epsilon)$ iterations. In this section, we show that an iterative approach can achieve the latter in $O(\log(1/\epsilon))$ iterations with an error-bounded precondition, avoiding the $\log(\kappa)$ factor in the number of iterations. We then use our approach to bound the bit complexity of solving a linear regression problem in input-sparsity time to high accuracy by using oblivious sketching approaches (see [19, 98]) that find a spectral approximation of the matrix.

**Remark 2.2.0.1.** Note that a bound on $\|x^{(k)} - x^*\|_{A^TA}$ does not imply a multiplicative error bound on $\|Ax - b\|_2$. It only gives an additive error bound of the following form.

$$\|Ax - b\|_2 \leq \|A(x - x^*)\|_2 + \|Ax^* - b\|_2 = \|x - x^*\|_{A^TA} + \|(I - A(A^TA)A^T)b\|_2.$$  

The second term of the right-hand side might be zero, in which case we have

$$\min_x \|Ax - b\|_2 = 0.$$  

However, it is not necessarily possible to achieve a zero error even if the optimal solution has zero error (at least not with numbers represented in fixed-point arithmetic).

**Lemma 2.2.0.2** (Bit complexity of Richardson’s iteration). Let $A \in \mathbb{R}^{n \times d}$ be a full-rank matrix, $n \geq d$. Let $\lambda \geq 1$, and $M, \tilde{M} \in \mathbb{R}^{d \times d}$ be symmetric matrices such that $A^TA \preceq M \preceq \lambda \cdot A^TA$ and $\|\tilde{M}^{-1} - M^{-1}\|_F \leq \frac{\epsilon}{\delta d \cdot \|A^TA\|_2}$. Let $x^{(k+1)} = x^{(k)} - \tilde{M}^{-1}(A^TAx^{(k)} - A^Tb)$. Then we have

$$\|x^{(k)} - x^*\|_M \leq \left(1 - \frac{1}{\lambda} + \epsilon\right)^k \|x^{(0)} - x^*\|_{M^*},$$

where $x^* = \arg\min_x \|Ax - b\|_2$.

**Proof.** We have $x^* = (A^TA)^{-1}A^Tb$, and

$$x^{(k+1)} - x^* = x^{(k)} - \tilde{M}^{-1}(A^TAx^{(k)} - A^Tb) - x^*$$

$$= x^{(k)} - \tilde{M}^{-1}(A^TAx^{(k)} - A^TAx^*) - x^*.$$
\[(I - \bar{M}^{-1}A^\top A)(x^{(k)} - x^*)\]

Therefore

\[
\|x^{(k+1)} - x^*\|_M = \|(I - \bar{M}^{-1}A^\top A)(x^{(k)} - x^*)\|_M
\]

\[
= \|(I - M^{-1}A^\top A + M^{-1}A^\top A - \bar{M}^{-1}A^\top A)(x^{(k)} - x^*)\|_M
\]

\[
\leq \|(I - M^{-1}A^\top A)(x^{(k)} - x^*)\|_M + \|(M^{-1}A^\top A - \bar{M}^{-1}A^\top A)(x^{(k)} - x^*)\|_M
\]

Now we have

\[
\|(I - M^{-1}A^\top A)(x^{(k)} - x^*)\|_M^2 = (x^{(k)} - x^*)^\top(I - A^\top AM^{-1})M(I - M^{-1}A^\top A)(x^{(k)} - x^*)
\]

Defining \(H = M^{-1/2}A^\top AM^{-1/2}\), we have

\[M^{1/2}(I - H)^2M^{1/2} = (I - A^\top AM^{-1})M(I - M^{-1}A^\top A).\]

Moreover,

\[
\frac{1}{\lambda}I = \frac{1}{\lambda}M^{-1/2}MM^{-1/2} \leq H \leq M^{-1/2}MM^{-1/2} = I.
\]

Therefore \(0 \leq I - H \leq (1 - \frac{1}{\lambda})I\), which implies

\[0 \leq (I - A^\top AM^{-1})M(I - M^{-1}A^\top A) \leq (1 - \frac{1}{\lambda})^2M.
\]

Hence

\[
\|(I - M^{-1}A^\top A)(x^{(k)} - x^*)\|_M^2 \leq (1 - \frac{1}{\lambda})^2\|(x^{(k)} - x^*)\|_M^2.
\]

Now we have

\[
\|(M^{-1}A^\top A - \bar{M}^{-1}A^\top A)(x^{(k)} - x^*)\|_M^2
\]
\[(x^{(k)} - x^*)^\top A^\top A (M^{-1} - \tilde{M}^{-1}) M (M^{-1} - \tilde{M}^{-1}) A^\top A (x^{(k)} - x^*).\]

Defining \(G = M^{1/2} \tilde{M}^{-1} M^{1/2}\), we have

\[(M^{-1} - \tilde{M}^{-1}) M (M^{-1} - \tilde{M}^{-1}) = M^{-1/2} (I - G)^2 M^{-1/2}\]

Now note that

\[I - G = I - M^{1/2} (\tilde{M}^{-1} - M^{-1} + M^{-1}) M^{1/2}\]
\[= I - M^{1/2} (\tilde{M}^{-1} - M^{-1}) M^{1/2} - M^{1/2} M^{-1} M^{1/2}\]
\[= -M^{1/2} (\tilde{M}^{-1} - M^{-1}) M^{1/2}\]

Therefore we have

\[
\|I - G\|_2 \leq \|I - G\|_F \leq \left\|M^{1/2}\right\|_F \left\|\tilde{M}^{-1} - M^{-1}\right\|_F
= \text{tr}(M) \left\|\tilde{M}^{-1} - M^{-1}\right\|_F \leq \frac{d \cdot \lambda \cdot \left\|A^\top A\right\|_2 \cdot \epsilon}{d \cdot \lambda \cdot \left\|A^\top A\right\|_2} = \epsilon
\]

Therefore we have

\[
\left\|\left(M^{-1} A^\top A - \tilde{M}^{-1} A^\top A\right) (x^{(k)} - x^*)\right\|_M^2
= (x^{(k)} - x^*)^\top A^\top A M^{-1/2} (I - G)^2 M^{-1/2} A^\top A (x^{(k)} - x^*)
= \left\|(I - G) M^{-1/2} A^\top A (x^{(k)} - x^*)\right\|_2^2
\leq \epsilon^2 \cdot \left\|M^{-1/2} A^\top A (x^{(k)} - x^*)\right\|_2^2
= \epsilon^2 \cdot (x^{(k)} - x^*)^\top A^\top A M^{-1} A^\top A (x^{(k)} - x^*)
\leq \epsilon^2 \cdot (x^{(k)} - x^*)^\top A^\top A M (x^{(k)} - x^*)
\leq \epsilon^2 \cdot \left\|x^{(k)} - x^*\right\|_M^2
\]

44
Therefore combining the above, we have
\[ \left\| x^{(k+1)} - x^* \right\|_M \leq \left( 1 - \frac{1}{\lambda} + \epsilon \right) \left\| x^{(k)} - x^* \right\|_M. \]

Remark 2.2.0.3. To guarantee that \( \| M^{-1} - \tilde{M}^{-1} \|_F \leq \frac{\epsilon}{d \cdot \| A^T A \|_2}, \) we require to use
\[ \log\left( \frac{d^3 \cdot \lambda \cdot \| A^T A \|_2 \| M^{-1} \|_2}{\epsilon} \right) \]
bits. Since \( A^T A \preceq M, \) we have \( (A^T A)^{-1} \succeq M^{-1}, \) which implies the about bound only requires \( \text{poly log}(d \lambda) \log(\frac{\epsilon}{\epsilon}) \) bit complexity. If we use \( M = A^T A, \) i.e., \( \lambda = 1, \) then picking \( \epsilon = \| b \|_2 \) and \( x^{(0)} = 0, \) we have
\[ \left\| x^{(1)} - x^* \right\|_{A^T A} \leq \epsilon, \]
since \( \| x^* \|_{A^T A} = \left\| A(A^T A)^{-1} A^T b \right\|_2 \leq \| b \|_2. \) This gives a bit complexity of \( \tilde{O}(\log(\frac{\kappa \| b \|_2}{\epsilon})) \) for the numbers we require for our computation. So assuming access to matrix \( \tilde{M}, \) we can find \( \tilde{x} \) such that
\[ \| A\tilde{x} - b \|_2 \leq \epsilon + \min_x \| A x - b \|_2, \]
in time \( \tilde{O}((\text{NNZ}(A) + d^2) \cdot \log(\frac{\kappa \| b \|_2}{\epsilon})). \)

Remark 2.2.0.4. If we use a spectral approximation, then in each iteration, we can only guarantee a decrease of \( (1 - \lambda^{-1}) \) factor in the error, and we have to perform \( \log(1/\epsilon) \) iterations of the Richardson’s iteration, to achieve the desired error. Therefore because in each iteration, the bit complexity of the solution vector is additively increased by \( O(\log \kappa), \) the bit complexity of the \( k \)’th iteration is \( O(k \cdot \log \kappa). \) This implies a running time of
\[ \tilde{O}((\text{NNZ}(A) + d^2) \cdot \log \kappa \cdot \log^2 \frac{1}{\epsilon}). \]
The following bounds the occurring error in computations we perform in each iteration of iterative algorithms for solving linear programs.

**Corollary 2.2.0.5.** Let $A \in \mathbb{R}^{n \times d}, W \in \mathbb{R}^{n \times n}$ be full-rank matrices and

$$u = \sqrt{WA}(A^TWA)^{-1}A^T\sqrt{W}g.$$

Moreover let $\lambda \geq 1$, and $M, \widetilde{M} \in \mathbb{R}^{d \times d}$ be symmetric matrices such that $A^TWA \preceq M \preceq \lambda \cdot A^TWA$ and $\left\|\widetilde{M}^{-1} - M^{-1}\right\|_F \leq \frac{\varepsilon}{\lambda \cdot \|A^TWA\|_2}$. Let $x^{(0)} = 0$,

$$x^{(k+1)} = x^{(k)} - \widetilde{M}^{-1}(A^TWAx^{(k)} - A^T\sqrt{W}g),$$

and $\bar{u} = \sqrt{WA}x^{(k)}$. Then we have

$$\|\bar{u} - u\|_2 \leq \lambda \cdot \left(1 - \frac{1}{\lambda} + \varepsilon\right)^k \|\sqrt{WA}(A^TWA)^{-1}A^T\sqrt{W}g\|_2.$$

**Proof.** Consider applying Richardson’s iteration to the following linear regression problem

$$\min_x \|\sqrt{WA}x - g\|_2$$

and suppose $x^* = \arg \min_x \|\sqrt{WA}x - g\|_2$. Then by Lemma 2.2.0.2, we have

$$\left\|x^{(k)} - x^*\right\|_M \leq \left(1 - \frac{1}{\lambda} + \varepsilon\right)^k \left\|x^{(0)} - x^*\right\|_M.$$

Therefore

$$\left\|x^{(k)} - x^*\right\|_{A^TWA} \leq \lambda \cdot \left(1 - \frac{1}{\lambda} + \varepsilon\right)^k \|x^*\|_{A^TWA}.$$

Now note that

$$\|x^*\|_{A^TWA}^2 = g^T\sqrt{WA}(A^TWA)^{-1}(A^TWA)(A^TWA)^{-1}A^T\sqrt{W}g$$

$$= \left\|\sqrt{WA}(A^TWA)^{-1}A^T\sqrt{W}g\right\|_2^2$$

$$\leq \|g\|_2^2,$$
where the inequality follows since \( \sqrt{W}A(A^TW)^{-1}A^T\sqrt{W} \) is a projection matrix. Moreover
\[
\left\| x^{(k)} - x^* \right\|_{A^TW} = \left\| \sqrt{W}A(x^{(k)} - x^*) \right\|_2 = \| \tilde{u} - u \|_2.
\]
Combining the above with the inequalities implies the result. □

Lemma 2.2.0.2 and Remark 2.2.0.3 give a bound on the bit complexity and running time of finding a solution assuming access to a matrix close enough to the inverse of \( A^TA \). However in many applications, we do not even have access to \( A^TA \). For example, in the case of input-sparsity time linear regression algorithm [19, 99] since computing \( A^TA \) takes \( \Omega(d \cdot \text{NNZ}(A)) \) arithmetic operations. Another recent example is for subquadratic time Kronecker regression algorithms since the size of \( A^TA \) is \( \Omega(d^2) \) and computing it prevents obtaining a subquadratic time algorithm.

Since Lemma 2.2.0.2 only needs a matrix that is close to the inverse of \( A^TA \), we instead find a spectral approximation \( M \) of \( A^TA \), and then using classic approaches [17, 105], we find a matrix \( \tilde{M} \) that is close to the inverse of \( M \), i.e., \( \left\| \tilde{M}^{-1} - M^{-1} \right\|_F \leq \frac{\varepsilon}{d \cdot \|A^TA\|_2^2} \). To find the spectral approximation we use the following result.

**Lemma 2.2.0.6 ([98, 106]).** Let \( A \in \mathbb{R}^{n \times d} \), \( 0 < \varepsilon, \delta < 1 \), and \( n \geq d \). There is an algorithm to sample a matrix \( S \) with \( O(d \log^8(d/\delta)/\varepsilon^2) \) rows, \( n \) columns, and \( s = \theta(\log^3(d/\delta)/\varepsilon) \) nonzero entries per columns, so that
\[
(1 - \varepsilon)A^TA \preceq A^TSA \preceq (1 + \varepsilon)A^TA,
\]
with probability at least \( 1 - \delta \). Moreover all entries of \( S \) are in \( \{0, \frac{1}{\sqrt{s}}, -\frac{1}{\sqrt{s}}\} \). Finally \( S \) can be sampled and multiplied with \( A \) in time
\[
O(\text{NNZ}(A) \cdot \text{poly log}(d/\delta)/\varepsilon^2).
\]

We are now equipped to prove our result for input-sparsity time linear regression.
Proof of Theorem 2.1.2. Our approach is to first compute a matrix $M$ such that $A^T A \preceq M \preceq \lambda \cdot A^T A$ for some constant $\lambda > 1$. We do this using Lemma 2.2.0.6. We pick $\delta = \text{poly}(n)$. We also pick $s$ to be a power of four, so the bit complexity of $S$ is controlled. Moreover we can pick a constant $\varepsilon_1$, in a way so that $\frac{1}{1-\varepsilon_1}$ is one plus a power of two. Then, we have

$$A^T A \preceq \frac{1}{1-\varepsilon_1} \cdot A^T S^T S A = M \preceq \frac{1+\varepsilon_1}{1-\varepsilon_1} \cdot A^T A = \lambda \cdot A^T A.$$  

Note that by our choice of parameters, the bit complexity of $M$ is equal to the bit complexity of $A$ up to constant and $\text{poly log}(nd)$ factors. Moreover, the condition number of $M$ is the same as $A$ (up to constant factors) since $M$ is spectrally close to $A^T A$. Note that by Lemma 2.2.0.6, we can compute $M$ in time $\tilde{O}(\text{NNZ}(A) + d^\omega)$ since the number of rows of $SA$ is $\tilde{O}(d)$.

We now can compute $\tilde{M}^{-1}$ such that $\left\| \tilde{M}^{-1} - M^{-1} \right\|_F \leq \frac{\varepsilon_2}{\lambda \cdot \|A\|_2}$, for some constant $0 < \varepsilon_2 < 1$ so that $1 - \frac{1}{\lambda} + \varepsilon_2 < 1$, in $\tilde{O}(d^\omega \cdot \log(\kappa))$ time using standard approaches [17, 105]. We then use Richardson’s iteration (Lemma 2.2.0.2) to compute a solution to the regression problem. The running time of this step is $\tilde{O}((\text{NNZ}(A) + d^2) \cdot \log \kappa \cdot \log^2(1/\varepsilon))$ according to Remark 2.2.0.4. This gives a total running time of

$$\tilde{O}((d^\omega + d^2 \cdot \log^2(1/\varepsilon) + \text{NNZ}(A) \cdot \log^2(1/\varepsilon)) \cdot \log \kappa).$$

\[\square\]

2.3 Inverse Maintenance Stability

In this section, we prove the backward stability of the inverse maintenance technique through the Sherman-Morrison-Woodbury identity (see Fact 2.1.2.1). Our formulation is based on the Frobenius norm error of the inverse matrix which in turn determines the bit complexity of the computed inverse. Note that due to the equivalence of norms, this provides bit complexity results in other norms up to polylogarithmic factors in dimension. We
show inverse maintenance is backward stable with a bit complexity of $\tilde{O}(\log(\kappa/\epsilon))$.

**Lemma 2.3.0.1** (Forward-backward error connection). Let $M, N \in \mathbb{R}^{n \times n}$ be invertible matrices and $\kappa > 1$ such that $\|N\|_F, \|N^{-1}\|_F \leq \kappa$. Suppose $\|M - N\|_F \leq \epsilon < \frac{1}{2\kappa}$. Then $\|M^{-1}\|_F \leq 2\kappa$, and

$$\|M^{-1} - N^{-1}\|_F \leq 2\kappa^2 \cdot \epsilon$$

*Proof.* Let $E = M - N$. Then by using the Woodbury identity we have

$$M^{-1} = N^{-1} - N^{-1}E(I + N^{-1}E)^{-1}N^{-1}$$

Now note that

$$(I + N^{-1}E)^{-1} = (I + N^{-1}M - I)^{-1} = M^{-1}N.$$  

Therefore

$$\|M^{-1}\|_F \leq \|N^{-1}\|_F + \|N^{-1}EM^{-1}\|_F \\
\leq \|N^{-1}\|_F + \|N^{-1}\|_F \|E\|_F \|M^{-1}\|_F \\
\leq \kappa + \kappa \epsilon \cdot \|M^{-1}\|_F.$$  

Therefore

$$\|M^{-1}\|_F \leq \frac{\kappa}{1 - \kappa \epsilon} \leq 2\kappa.$$  

Hence,

$$\|M^{-1} - N^{-1}\|_F = \|N^{-1}EM^{-1}\|_F \leq \|N^{-1}\|_F \|E\|_F \|M^{-1}\|_F \leq 2\kappa^2 \epsilon.$$

We are now equipped to analyze the stability of the Sherman-Morrison-Woodbury formula for inverse maintenance. Note that there are two sources of error in this formulation.
One is from the inverse of the original matrix, and the other is from the inverse of the smaller matrix we need to compute to make the low-rank update to the inverse.

**Lemma 2.1.2.2 (Backward Stability of Inverse Maintenance).** Let $Z \in \mathbb{R}^{n \times n}$, $\tilde{Z} \in \mathbb{R}^{n \times n}$, $C \in \mathbb{R}^{m \times m}$ be invertible matrices. Moreover let $U, V \in \mathbb{R}^{n \times m}$ such that $Z + UCV^\top$ is invertible.

Let $\kappa > m + n$ such that

$$
\|U\|_F, \|V\|_F, \|C\|_F, \|C^{-1}\|_F, \|Z\|_F, \|Z^{-1}\|_F, \|Z + UCV^\top\|_F, \|(Z + UCV^\top)^{-1}\|_F \leq \kappa
$$

and $0 < \epsilon_1, \epsilon_2 < 1$. Suppose

$$
\|\tilde{Z} - Z\|_F \leq \epsilon_1. \tag{2.1.2}
$$

If $D \in \mathbb{R}^{m \times m}$ is an invertible matrix such that

$$
\left\| D^{-1} - (C^{-1} + V^\top \tilde{Z}^{-1} U)^{-1} \right\|_F \leq \epsilon_2,
$$

then

$$
\left\| (\tilde{Z}^{-1} - \tilde{Z}^{-1} UD^{-1} V^\top \tilde{Z}^{-1})^{-1} - (Z + UCV^\top) \right\|_F \leq 512 \kappa^{26} \epsilon_2 + \epsilon_1.
$$

**Proof.** By triangle inequality and (Equation 2.1.2), we have

$$
\left\| (\tilde{Z}^{-1} - \tilde{Z}^{-1} UD^{-1} V^\top \tilde{Z}^{-1})^{-1} - (Z + UCV^\top) \right\|_F \leq \left\| (\tilde{Z}^{-1} - \tilde{Z}^{-1} UD^{-1} V^\top \tilde{Z}^{-1})^{-1} - (Z + UCV^\top) \right\|_F + \|\tilde{Z} - Z\|_F
$$

$$
\leq \left\| (\tilde{Z}^{-1} - \tilde{Z}^{-1} UD^{-1} V^\top \tilde{Z}^{-1})^{-1} - (Z + UCV^\top) \right\|_F + \epsilon_1. \tag{2.3.1}
$$

Note that $S := C^{-1} + V^\top \tilde{Z}^{-1} U$ is a Schur complement of the following matrix.

$$
T := \begin{bmatrix} C^{-1} & V^\top \\ U & -\tilde{Z} \end{bmatrix} = \begin{bmatrix} I & 0 \\ UC & I \end{bmatrix} \begin{bmatrix} C^{-1} & 0 \\ 0 & -\tilde{Z} - UCV^\top \end{bmatrix} \begin{bmatrix} I & CV^\top \\ 0 & I \end{bmatrix}
$$
Note that since $C^{-1}$ and $\bar{Z} - UCV^T$ (the Schur complement) are invertible, $T$ is invertible and

$$T^{-1} = \begin{bmatrix} I & -CV^T \\ 0 & I \end{bmatrix} \begin{bmatrix} C^{-1} & 0 \\ 0 & -\bar{Z} - UCV^T \end{bmatrix}^{-1} \begin{bmatrix} I \\ 0 \end{bmatrix}.$$ 

Since $S$ is the Schur complement of $T$,

$$\|S^{-1}\|_F \leq \|T^{-1}\|_F \leq \left( \sqrt{n+m+\|UC\|_F} \right) \left( \sqrt{n+m+\|CV^T\|_F} \right) \left( \|C\|_F + \|\bar{Z} + UCU^T\|_F^{-1} \right) \leq 2(m+n+\kappa^4)\kappa^2 \leq 4\kappa^6. \quad (2.3.2)$$

Moreover

$$\|S\|_F \leq \|C^{-1}\|_F + \|V^T\|_F \|\bar{Z}^{-1}\|_F \|U\|_F \leq 2\kappa^3.$$ 

Therefore since $\|D^{-1} - (C^{-1} + V^T\bar{Z}^{-1}U)^{-1}\|_F \leq \varepsilon_2$, by Lemma 2.3.0.1

$$\|D - (C^{-1} + V^T\bar{Z}^{-1}U)\|_F \leq (4\kappa^6)^2\varepsilon_2 = 16\kappa^{12}\varepsilon_2. \quad (2.3.3)$$

Now let

$$M := \begin{bmatrix} D - V^T\bar{Z}^{-1}U & V^T \\ U & -\bar{Z} \end{bmatrix}.$$ 

Then by (Equation 2.3.3),

$$\|M - T\|_F \leq 16\kappa^{12}\varepsilon_2. \quad (2.3.4)$$

Moreover by triangle inequality $\|T\|_F \leq \|C^{-1}\|_F + \|U\|_F + \|V\|_F + \|\bar{Z}\|_F \leq 4\kappa$. Combining this with (Equation 2.3.2) and (Equation 2.3.4), noting that $M^{-1}$ is invertible since $-\bar{Z}$ and $D - V^T\bar{Z}^{-1}U + V^T\bar{Z}^{-1}U = D$ (the Schur complement) are invertible, and using Lemma 2.3.0.1, we have

$$\|M^{-1} - T^{-1}\| \leq (4\kappa^6)^2 \cdot 16\kappa^{12}\varepsilon_2 = 256\kappa^{24}\varepsilon_2.$$
Now note that $-\tilde{Z} - UCV^T$ and $-\tilde{Z} - U(D - V^T\tilde{Z}^{-1}U)^{-1}V^T$ are the Schur complements (of the corresponding block) of $T$ and $M$, respectively. Therefore

$$\left\|\left(\tilde{Z} + U(D - V^T\tilde{Z}^{-1}U)^{-1}V^T\right)^{-1} - \left(\tilde{Z} + UCV^T\right)^{-1}\right\|_F \leq \|M^{-1} - T^{-1}\|_F \leq 256\kappa^24\epsilon_2.$$  

By Woodbury identity,

$$(\tilde{Z} + U(D - V^T\tilde{Z}^{-1}U)^{-1}V^T)^{-1} = \tilde{Z}^{-1} - \tilde{Z}^{-1}U(D - V^T\tilde{Z}^{-1}U + V^T\tilde{Z}^{-1}U)^{-1}V^T\tilde{Z}^{-1}$$

$$= \tilde{Z}^{-1} - \tilde{Z}^{-1}UD^{-1}V^T\tilde{Z}^{-1}.$$  

Therefore

$$\left\|\left(\tilde{Z}^{-1} - \tilde{Z}^{-1}UD^{-1}V^T\tilde{Z}^{-1}\right) - \left(\tilde{Z} + UCV^T\right)^{-1}\right\|_F \leq 256\kappa^24\epsilon_2.$$  

Thus since $\|Z + UCV^T\|_F, \|(Z + UCV^T)^{-1}\|_F \leq \kappa$, by Lemma 2.3.0.1,

$$\left\|\left(\tilde{Z}^{-1} - \tilde{Z}^{-1}UD^{-1}V^T\tilde{Z}^{-1}\right) - \left(\tilde{Z} + UCV^T\right)^{-1}\right\|_F \leq (2\kappa^2) \cdot 256\kappa^24\epsilon_2 = 512\kappa^26\epsilon_2.$$  

The result follows from combining this with (Equation 2.3.1). □

We proved the stability of the inverse maintenance process in this section. Equipped with this, we next develop inverse maintenance data structures for both sparse and dense matrices. We later use these data structures for iterative algorithms for solving $p$-norm minimization and LP.

2.3.1 Bit Complexity of Dense Inverse Maintenance Data Structure

In this section, we present our data structures for dense matrices. For dense matrices, our data structure only relies on the stability of inverse maintenance proven in Lemma 2.1.2.2.

**Theorem 2.3.1.1** (Dense data structure — DDS). Let $n \geq d$. There exists a data structure with the following operations that maintain an explicit matrix $Z^{-1}$. 


**Algorithm 1: Dense inverse maintenance data structure (DDS)**

1. **Variables:** $n, d \in \mathbb{N}$ with $n > d$, $A \in \mathbb{R}^{n \times d}$, $Z^{-1} \in \mathbb{R}^{d \times d}$, $v \in \mathbb{R}^n$, $1 > \varepsilon > 0$.

2. **Procedure** `INITIALIZE($A \in \mathbb{R}^{n \times d}, w \in \mathbb{R}^n, \varepsilon$)

   3. Set $A$, $\varepsilon$, $n$, and $d$ according to the input

   4. Set $v \leftarrow w$

   5. Compute the matrix $Z^{-1}$ such that $\|Z - A^T VA\|_F \leq \varepsilon$  // for example using the approach of [17, 105]

3. **Procedure** `UPDATE($S \subseteq [n], u \in \mathbb{R}^{|S|}$)

   6. Set $q_S = u - v_S$ where $q \in \mathbb{R}^n$

   7. Set $v_S = u$

   8. Update $Z^{-1}$ using Woodbury identity as the following

   9. $Z^{-1} \leftarrow Z^{-1} - Z^{-1} A_S^\top D^{-1} A_S Z^{-1}$, where $D^{-1} \in \mathbb{R}^{|S| \times |S|}$ is a matrix such that

      $\left\|D^{-1} - (Q_{S,S}^{-1} + A_S Z^{-1} A_S^\top)^{-1}\right\|_F \leq \varepsilon$, and $Q$ is the diagonal matrix corresponding to $q$.

   10. Round entries of $Z^{-1}$ to numbers with $\lceil \log(10d^2 \cdot k/\varepsilon) \rceil$ bits.

4. **Procedure** `QUERY($\mathbf{b} \in \mathbb{R}^d$)

   11. Compute and return $Z^{-1} \mathbf{b}$.

---

- **INITIALIZE($A \in \mathbb{R}^{n \times d}, w \in \mathbb{R}^n, \varepsilon$):** Sets $v = w$, and computes a linear operation $Z^{-1}$ such that

  $$\|Z - A^T VA\|_F \leq \varepsilon,$$

  where $V$ is the diagonal matrix corresponding to $v$.

- **UPDATE($S \subseteq [n], u \in \mathbb{R}^{|S|}$):** Sets $v_S = u$ and updates $Z^{-1}$ such that after $k$ calls to UPDATE, we have

  $$\|Z - A^T VA\|_F \leq (512 \cdot k \cdot \kappa^{26} + 2 \cdot k \cdot \kappa^2 + 1) \cdot \varepsilon,$$

  where $\kappa > n + d$ is a number such that

  $$\|A\|_F, \|Q_{S,S}\|_F, \|Q_{S,S}^{-1}\|_F, \|A^T VA\|_F, \|(A^T VA)^{-1}\|_F \leq \kappa,$$
during all the updates and
\[
\varepsilon < \frac{1}{2k \cdot (512 \cdot (k + 1) \cdot \kappa^{26} + 2 \cdot k \cdot \kappa^2 + 1)}.
\]

- **QUERY** (\(\vec{b} \in \mathbb{R}^d\)): Computes and returns \(Z^{-1}\vec{b}\).

The running time of initialize, update, and query are \(\text{MM}(n, d, d) \cdot O(\log(\kappa/\varepsilon))\),

\[
(\text{MM}(d, |S|, |S|) + \text{MM}(d, d, |S|) + |S|^{\omega}) \cdot O(\log(\kappa/\varepsilon)),
\]

and \(d^2 \cdot O(\log(\kappa/\varepsilon))\), respectively. Moreover the bit complexity of the matrix \(Z^{-1}\) is \(\tilde{O}(\log(\kappa/\varepsilon))\).

**Proof.** We show that the data structure in Algorithm 1 achieves the desired result. Computing the matrix \(A^\top VA\) takes \(\text{MM}(n, d, d) \cdot O(\log(\kappa))\) time, and computing the matrix \(Z^{-1}\) such that \(\|Z - A^\top VA\|_F \leq \varepsilon\) takes \(d^\omega \cdot O(\log(\kappa/\varepsilon))\) takes. This gives the bound on the bit complexity of initialization follows from \(d \leq n\).

Now note that at the end of each update procedure, we round the matrix to numbers with \(\tilde{O}(\log(\kappa/\varepsilon))\) bits. For update we need to compute the matrix \(D\) such that

\[
\left\|D^{-1} - (Q^{S, S}_{S} + A_{S}Z^{-1}A_{S}^\top)^{-1}\right\|_F \leq \varepsilon
\]

which can be done in time \((\text{MM}(|S|, d, d) + \text{MM}(|S|, |S|) + |S|^{\omega}) \cdot O(\log(\kappa/\varepsilon))\). Given the matrix \(D^{-1}\), updating the matrix \(Z^{-1}\) can be done in time

\[
(\text{MM}(d, |S|, |S|) + \text{MM}(d, d, |S|)) \cdot O(\log(\kappa/\varepsilon)).
\]

The cost of the query is a matrix-vector multiplication which is \(d^2 \cdot O(\log(\kappa/\varepsilon))\).

We now need to bound the error of our matrix after \(k\) updates. We prove this by induction. The base case trivially follows from the condition in the initialization. Now suppose
after $k$ updates

$$\|Z - A^TVA\|_F \leq (512 \cdot k \cdot \kappa^{26} + 2 \cdot k \cdot \kappa^2 + 1) \cdot \varepsilon,$$

and $Q$ is the diagonal matrix corresponding to the $k+1$’st update. Let

$$\tilde{Z}^{-1} = Z^{-1} - Z^{-1}A_S^TA_SZ^{-1}, \text{ with } \left\|D^{-1} - (Q_{SS}^{-1} + A_SZ^{-1}A_S^T)^{-1}\right\|_F \leq \varepsilon,$$

and $\tilde{Z}^{-1}$ is obtained by rounding the entries of $\tilde{Z}^{-1}$ to numbers with $\lceil \log(10d^2 \cdot \kappa/\varepsilon) \rceil$ bits. Lemma 2.1.2.2 directly gives

$$\left\|\tilde{Z} - A^T(V + Q)A\right\|_F \leq (512 \cdot (k + 1) \cdot \kappa^{26} + 2 \cdot k \cdot \kappa^2 + 1) \cdot \varepsilon.$$

Moreover since by assumption $\|A^T(V + Q)A\|_F, \|(A^T(V + Q)A)^{-1}\|_F \leq \kappa$, and

$$\left\|\tilde{Z} - A^T(V + Q)A\right\|_F \leq (512 \cdot (k + 1) \cdot \kappa^{26} + 2 \cdot k \cdot \kappa^2 + 1) \cdot \varepsilon < \frac{1}{\frac{1}{2} \kappa},$$

by Lemma 2.3.0.1, we have $\left\|\tilde{Z}^{-1}\right\|_F \leq 2\kappa$. Therefore the number of bits before decimal for the entries $\tilde{Z}^{-1}$ is bounded by $\log_2(2\kappa) + 1$. Therefore the rounding only introduces error in the bits after the decimal point. Therefore

$$\left\|\tilde{Z}^{-1} - Z^{-1}\right\| \leq \varepsilon.$$

Invoking Lemma 2.3.0.1 and because $\varepsilon < \frac{1}{4\kappa}$ (by assumption), $\left\|\tilde{Z}^{-1}\right\|_F \leq 2\kappa$, and

$$\left\|\tilde{Z}\right\|_F \leq \left\|\tilde{Z} - A^T(V + Q)A\right\|_F + \|A^T(V + Q)A\|_F \leq \frac{1}{2\kappa} + \kappa \leq 2\kappa,$$

we have $\left\|\tilde{Z} - Z\right\|_F \leq 2\kappa^2 \cdot \varepsilon$. Therefore by triangle inequality,

$$\left\|\tilde{Z} - A^T(V + Q)A\right\|_F \leq \left\|\tilde{Z} - Z\right\|_F + \left\|\tilde{Z} - A^T(V + Q)A\right\|_F.$$
\[ \leq (512 \cdot (k + 1) \cdot \kappa^{26} + 2 \cdot (k + 1)\kappa^2 + 1) \cdot \varepsilon. \]

Therefore the desired error bound holds. \( \square \)

**Remark 2.3.1.2.** Note that if we do not perform the rounding step in the **UPDATE** procedure of DDS, after \( k \) updates, the bit complexity increases by a factor of \( k \) since the update involves multiplying matrices. However, because of the upper bound assumption on \( \varepsilon \), after the rounding, the bit complexity of the resulting matrix is \( \tilde{O}(\log(k \cdot \kappa/\varepsilon)) \). As we see later the number of iterations in our algorithms is of the form \( \text{poly}(n) \log(f(n)) \), where \( f(n) \) is at most an exponential function in \( n \) (e.g., for LPs, the number of iterations is \( \tilde{O}(n^{0.5} \log(\frac{R}{\varepsilon})) \)). Therefore \( \tilde{O}(\log(k \cdot \kappa/\varepsilon)) \) in our algorithms is \( \tilde{O}(\log(k/\varepsilon)) \).

**Remark 2.3.1.3.** Theorem 2.3.1.1 requires \( \|Q_{S,S}\|_F, \|Q_{S,S}^{-1}\|_F \leq \kappa \), where \( Q \) is the diagonal matrix corresponding to the difference of weights from one iteration to the next. Although we usually update the inverse only for weights that have changed significantly in our algorithms, even if the difference is small, this condition does not impose a limitation on our data structure since we can implement each update as two updates. For example, if \( q_i > 0 \) is small, we first make an update with \( \tilde{q}_i^{(1)} = q_i + 1 \) and then we make a second update with \( \tilde{q}_i^{(2)} = 1 \). Note that when doing this a number should be added and subtracted so that the Frobenius norm of \( A^T V A \) and \( (A^T V A)^{-1} \) also stay small when we perform the auxiliary update step.

### 2.3.2 Bit Complexity of Sparse Inverse Maintenance Data Structure

We now turn to the sparse case. We first give an algorithm and bit complexity bounds for computing a linear operator for the inverse of a sparse matrix based on the block-Krylov approach [107, 108]. Our algorithm builds on [45] but has some differences from the algorithm presented in that work since we use the improved analysis presented in [46].

**Theorem 2.3.2.1** ([45, 46]). Given a sparse \( n \times d \) matrix \( A \) with max entry-wise magnitude at most 1, a diagonal \( n \times n \) matrix \( W \) with entry-wise magnitude at most 1 and \( m \leq \)
\(d^{(\omega-2)/(\omega-1)}\), along with \(\kappa\) that upper bounds the condition numbers of \(A\) and \(W\), with probability at least \(1 - n^{-20}\), we can obtain in time

\[
\tilde{O}\left(\left(d \cdot \text{NNZ}(A) \cdot m + \left(\frac{d}{m}\right)^{\omega} m^2 \right) \log (\kappa)\right)
\]

a linear operator \(Z_{A^\top W A}\) such that

\[
\left\|Z_{A^\top W A} - (A^\top W A)^{-1}\right\|_F \leq \kappa^{-10} n^{-10}.
\]

Moreover, for a \(d \times r\) matrix \(B\), where \(r \leq d/m\), \(Z_{A^\top W A} B\) can be computed in time \(\tilde{O}((r \cdot \text{NNZ}(A) \cdot m^2 + d^2 r^{\omega-2} \log(\kappa/\epsilon))\).

**Proof.** The statements below are closely based on the top-level claims in [45]{#3}. Since \(A^\top W A\) is already symmetrized, we can ignore the outer step involving a multiplication by the transpose of an asymmetric matrix. So we will show how to give access to an operator \(Z_{A^\top W A}\) such that

\[
\left\|Z_{A^\top W A} - (A^\top W A)^{-1}\right\|_F \leq \epsilon
\]

(2.3.5)

The algorithm that computes access to this \(Z\) was given in Section 7 of [45].

1. Perturb with random Gaussian \(R\) to form the perturbed matrix

\[
\hat{A} = A^\top W A + R
\]

2. Generate Krylov space with \(\tilde{O}(m)\) extra columns,

\[
K = \left[ G^s \quad \hat{A} G^s \quad \hat{A}^2 G^s \quad \ldots \quad \hat{A}^{m-1} G^s \right],
\]

where $s = d/m$, and $G^s$ is a sparse Gaussian matrix with $s$ columns and $\text{NNZ}(G^s) = \tilde{O}(n)$.

3. Compute the inverse of the block-Hankel matrix $H = K^T \hat{A} K$.

Specifically, for step (3), the $Z_H$ generated by the block-Hankel solver is the product of two explicit matrices, each with $\tilde{O}(m \log \kappa)$ bits,

$$Z_H = X_H Y_H^T$$

such that the cost of computing $X_H B$, $Y_H B$, $X_H^T B$, $Y_H^T B$ for some $d$-by-$r$ matrix $B$ with up to $\tilde{O}(m \log \kappa)$ bits per entry is $\tilde{O}(m^2 \cdot \text{MM}(\frac{d}{m}, \frac{d}{m}, \frac{d}{m}, r) \cdot \log \kappa)$ by Lemma 6.6 of [45]^1. The cost of multiplying $Z_{A^TWA}$ against a $d$-by-$r$ matrix $B$ is then broken down into three parts:

1. The cost of performing the multiplication $K^T B$ which takes $\tilde{O}(\text{NNZ}(A) \cdot m^2 \cdot r \cdot \log^2(\kappa/\varepsilon))$ time.

2. The cost of multiplying $Y_H^T$ and $X_H$ against a $d$-by-$r$ matrix: by Lemma 6.6 of [45]^1, this takes time $\tilde{O}(\text{MM}(\frac{d}{m}, \frac{d}{m}, r) \cdot m^2 \log \kappa)$ by performing fast Fourier transform on the blocks of $X_H$ and $Y_H$.

3. The cost of multiplying $K$ with a $d$-by-$r$ matrix from right which takes $\tilde{O}(\text{NNZ}(A) \cdot m^2 \cdot r \cdot \log^2(\kappa/\varepsilon))$ time.

\[\square\]

We are now equipped to present our sparse data structure and bound the bit complexity of its operations.

**Theorem 2.3.2.2** (Sparse data structure — SDS). Let $n \geq d$ and $m \leq n^{1/4}$. There exists a data structure with the following operations that maintain an inverse operator as the sum of an explicit matrix $T$ and a block-Krylov-based inverse (as represented in Theorem 2.3.2.1).
Algorithm 2: Sparse inverse maintenance data structure (SDS)

1 Variables: $n, d, m \in \mathbb{N}$ with $n > d, m \leq n^{0.25}, A \in \mathbb{R}^{n \times d}$, linear operator $Z \in \mathbb{R}^{d \times d}$, explicit matrix $T \in \mathbb{R}^{d \times d}$, $v \in \mathbb{R}^n$, $1 > \varepsilon > 0$.

2 Procedure INITIALIZE($A \in \mathbb{R}^{n \times d}, w \in \mathbb{R}^n, \varepsilon$)

3 Set $v \leftarrow w$

4 Compute the linear operator $Z$ such that $\|Z^{-1} - A^T VA\|_F \leq \varepsilon$ // using Theorem 2.3.2.1 and setting the error bounds small enough according to Lemma 2.3.0.1

5 Set $T$ to the matrix of all zeros

6 Procedure UPDATE($S \subseteq [n], u \in \mathbb{R}^{|S|})$

7 if $|S| \geq \frac{n}{m}$ then

8 Set $w \in \mathbb{R}^n$ to a vector with $w_i = u_i$, if $i \in S$, and $w_i = v_i$, if $i \in [n] \setminus S$.

9 INITIALIZE($A, w, \varepsilon$)

10 else

11 Set $q_S = u - v_S$ where $q \in \mathbb{R}^n$

12 Set $v_S = u$

13 Update the matrix $T$ as the following

\[
T \leftarrow T - (Z + T)^T A_S^T D^{-1} A_S (Z + T),
\]

(2.3.6)

where $D^{-1} \in \mathbb{R}^{|S| \times |S|}$ is a matrix such that

\[
\|D^{-1} - (Q^{-1}_S + A_S (Z + T) A_S^T)^{-1}\|_F \leq \varepsilon,
\]

and $Q$ is the diagonal matrix corresponding to $q$.

14 Round entries of $T$ to numbers with $\lceil \log(10 d^2 \cdot k/\varepsilon) \rceil$ bits.

16 Procedure QUERY($\bar{b} \in \mathbb{R}^d$)

17 Compute and return $Z \bar{b} + T \bar{b}$. // $Z \bar{b}$ is computed according to Theorem 2.3.2.1

- **INITIALIZE($A \in \mathbb{R}^{n \times d}, w \in \mathbb{R}^n, \varepsilon$):** Sets $v = w$, and initializes the explicit matrix $T \in \mathbb{R}^{d \times d}$ and a linear operator $Z$ (see Theorem 2.3.2.1) such that

\[
\| (Z + T)^{-1} - A^T VA \|_F \leq \varepsilon,
\]

where $V$ is the diagonal matrix corresponding to $v$.

- **UPDATE($S \subseteq [n], u \in \mathbb{R}^{|S|}$):** Sets $v_S = u$ and updates $Z$ and $T$ such that after $k$ calls
to update, we have

\[ \| (Z + T)^{-1} - A^\top VA \|_F \leq (512 \cdot k \cdot k^2 + 2 \cdot k \cdot k^2 + 1) \cdot \varepsilon, \]

where \( \kappa > n + d \) is a number such that

\[ \|A\|_F, \|Q_{S,S}\|_F, \|Q_{S,s}\|_F, \|A^\top VA\|_F, \|(A^\top VA)^{-1}\|_F \leq \kappa, \]

during all the updates and

\[ \varepsilon < \frac{1}{2\kappa \cdot (512 \cdot (k + 1) \cdot k^2 + 2 \cdot k \cdot k^2 + 1)}. \]

- **QUERY(\( \overline{\mathbf{b}} \))**: Computes and returns \((Z + T)\overline{\mathbf{b}}\).

The running time of initialize, and query are \( \tilde{O}\left( (d \cdot \text{NNZ}(A) \cdot m + \left( \frac{d}{m} \right)^2 m^2 \log^2 (\kappa / \varepsilon) \right), \right) \)

and \( \tilde{O}(\text{NNZ}(A) \cdot m^2 + d^2 \cdot \log^2 (\kappa / \varepsilon)) \), respectively. The running time of updates is equal to initialize if \(|S| \geq n/m\), and is equal to

\[ \left( \text{NNZ}(A) \cdot m^2 \cdot |S| + d^2 \cdot |S|^{\omega^2 - 2} + \text{MM}(d, |S|, |S|) + |S|^{\omega} \right) \cdot O(\log^2 (\kappa / \varepsilon)), \]

otherwise.

**Proof.** We show that the data structure in Algorithm 2 achieves the desired result. The running time for the initialization and update when \(|S| \geq n/m\) follow directly from Theorem 2.3.2.1. The running time of query follows by invoking the second part of Theorem 2.3.2.1 for a matrix with one column.

For updates with \(|S| < n/m\), we first need to compute \(Z^\top A_{S}^\top\) which by Theorem 2.3.2.1 can be done in time \( \tilde{O}( (\text{NNZ}(A) \cdot m^2 \cdot |S| + d^2 \cdot |S|^{\omega^2 - 2}) \log^2 (\kappa / \varepsilon) ) \). After this multiplication, the number of bits of the resulting matrix can be reduced to \( \tilde{O}(\log(\kappa / \varepsilon)) \) because the condition numbers of \( A^\top VA \) is bounded by \( \kappa^{O(1)} \).
ror can be counted as the error of the linear operator of the inverse. Then with an ex-
tra cost of \( \tilde{O}(d^2 \log(\kappa/\epsilon)) \), we can compute \((Z + T)^\top A_S^\top \). Therefore \(A_S(Z + T) A_S^\top \) can be computed in time \( \tilde{O}(MM(|S|, d, |S|) \cdot \log(\kappa/\epsilon)) \). Now \(D^{-1}\) can be computed in time \( \tilde{O}(|S|^\omega \cdot \log(\kappa/\epsilon)) \). Finally since we already have computed \((Z + T)^\top A_S^\top\), \(T\) can be updated in time \( \tilde{O}((MM(d, |S|, |S|) + MM(d, d, |S|)) \cdot \log(\kappa/\epsilon)) \).

We now need to bound the error of our matrix after \( k \) updates. We prove this by induction. The base case trivially follows from the condition in the initialization and Lemma 2.3.0.1. Now suppose after \( k \) updates

\[
\| (Z + T)^{-1} - A^\top VA \|_F \leq (512 \cdot k \cdot \kappa^{26} + 2 \cdot k \cdot \kappa^2 + 1) \cdot \epsilon,
\]

and \(Q\) is the diagonal matrix corresponding to the \( k + 1 \)'st update. Let

\[
\tilde{T} = T - (Z + T)^\top A_S^\top D^{-1} A_S(Z + T), \quad \text{with} \quad \| D^{-1} - (Q_{S,S}^{-1} + A_S(Z + T) A_S^\top)^{-1} \|_F \leq \epsilon,
\]

and \(\tilde{T}\) is the matrix obtained by rounding the entries of \(\tilde{T}\) to numbers with \(\lceil \log(10d^2 \cdot \kappa/\epsilon) \rceil \) bits. Also let \(\tilde{Z} := Z + \tilde{T}\) and \(\tilde{Z} := Z + \tilde{T}\). Lemma 2.1.2.2 directly gives

\[
\| \tilde{Z}^{-1} - A^\top (V + Q) A \|_F \leq (512 \cdot (k + 1) \cdot \kappa^{26} + 2 \cdot k \cdot \kappa^2 + 1) \cdot \epsilon.
\]

Moreover since by assumption \(\|A^\top (V + Q) A\|_F, \| (A^\top (V + Q) A)^{-1} \|_F \leq \kappa\), and

\[
\| \tilde{Z}^{-1} - A^\top (V + Q) A \|_F \leq (512 \cdot (k + 1) \cdot \kappa^{26} + 2 \cdot k \cdot \kappa^2 + 1) \cdot \epsilon < \frac{1}{2\kappa},
\]

by Lemma 2.3.0.1, we have \(\|\tilde{Z}\|_F \leq 2\kappa\). Therefore the number of bits before decimal for the entries \(\tilde{Z}\) is bounded by \(\log_2(2\kappa) + 1\). Therefore the rounding only introduces error in the bits after the decimal point. Therefore

\[
\| \tilde{Z} - \tilde{Z} \|_F = \| \tilde{T} - \tilde{T} \|_F \leq \epsilon.
\]
Invoking Lemma 2.3.0.1 and because $\varepsilon < \frac{1}{4\kappa}$ (by assumption), $\|Z\|_F \leq 2\kappa$, and

$$\|Z^{-1}\|_F \leq \|Z^{-1} - A^T (V + Q)A\|_F + \|A^T (V + Q)A\|_F \leq \frac{1}{2\kappa} + \kappa \leq 2\kappa,$$

we have $\|Z^{-1} - \tilde{Z}^{-1}\|_F \leq 2\kappa^2 \cdot \varepsilon$. Therefore by triangle inequality,

$$\|Z^{-1} - A^T (V + Q)A\|_F \leq \|Z^{-1} - \tilde{Z}^{-1}\|_F + \|\tilde{Z}^{-1} - A^T (V + Q)A\|_F \leq (512 \cdot (k + 1) \cdot \kappa^{26} + 2 \cdot (k + 1)\kappa^2 + 1) \cdot \varepsilon.$$

Therefore the desired error bound holds. \[\Box\]

2.4 Linear Programming Using Interior Point Methods (IPM)

In this section, we consider linear programming problems of the following form.

$$\min_{x: A^T x = b, x \geq 0} c^T x,$$

where $A \in \mathbb{R}^{n \times d}$, $b \in \mathbb{R}^d$ and $c \in \mathbb{R}^n$. We consider a variety of interior point methods for this problem. Our main result is the following that bounds the bit complexity of the algorithm of [7], which is the derandomized version of [6] — see Section 2.4.2. A main difference between our algorithm and that of [6, 7] is the choice of initial feasible solution. Inspired by [100] and in contrast with [6, 7], we select the initial feasible solution so that the condition number and $R/r$ stay the same up to polynomial factors. Recall $r$ and $R$ are the inner and outer radius of the LP (see Definition 2.1.1.3).

**Theorem 2.1.1.4 (Robust IPM).** Given $A \in \mathbb{R}^{n \times d}$ with full column-rank and condition number $\kappa$, $b \in \mathbb{R}^d$, $c \in \mathbb{R}^n$ all with bit complexity of $\log(\kappa)$, and an error parameter $0 < \varepsilon < 1$, suppose the inner radius and outer radius of the linear program $\min_{A^T x = b, x \geq 0} c^T x$ is
Then there is an algorithm that computes $\hat{x} \in \mathbb{R}^n$ such that

$$c^T\hat{x} \leq \min_{A^Tx = b, x \geq 0} c^T x + \epsilon, \quad \text{and} \quad \|A^T\hat{x} - b\|_2 \leq \epsilon,$$

in time $\tilde{O}\left(\left(n^\omega + n^{2.5-\alpha/2} + n^{2+1/6}\right) \cdot \log\left(\frac{sR}{\epsilon r}\right) \cdot \log\left(\frac{R}{\epsilon r}\right)\right)$.

As discussed in Section 2.1.3, $\log(\kappa)$ can be $\Omega(n)$ even for matrices with bit complexity $O(1)$. Moreover, as we discuss in Section 2.4.1, $\log(R/r)$ can be $\Omega(n)$ as well. This gives a total running time of $O(n^{\omega+2})$ for algorithms of [6, 7]. Note that there are instances in which $\log(R/r) = O(1)$ while $\kappa = \Omega(n)$. Motivated by this, we present the following algorithm based on solving linear systems using shifted numbers [18] that replaces the $\log(\kappa)$ factor with $n^{0.5}$. In instances with $\log(R/r) = O(1)$, $\kappa = \Omega(n)$, this approach is faster than Theorem 2.1.1.4 by a factor of $n^{0.5}$.

**Theorem 2.1.1.5 (Inverse-free IPM).** Given $A \in \mathbb{R}^{n \times d}$ with full column-rank, $b \in \mathbb{R}^d$, $c \in \mathbb{R}^n$ all with bit complexity of $\ell$, and an error parameter $0 < \epsilon < 1$, suppose the inner radius and outer radius of the linear program $\min_{A^Tx = b, x \geq 0} c^T x$ is $r$ and $R$, respectively. Then there is an algorithm that finds $\hat{x} \in \mathbb{R}^n$ such that

$$c^T\hat{x} \leq \min_{A^Tx = b, x \geq 0} c^T x + \epsilon, \quad \text{and} \quad \|A^T\hat{x} - b\|_2 \leq \epsilon,$$

in time $\tilde{O}\left(n^{\omega+0.5} \cdot (\ell + \log\left(\frac{R}{\epsilon r}\right)) \cdot \log\left(\frac{R}{\epsilon r}\right)\right)$.

We use the classic IPM that uses the 2-norm as its potential function for the above result. A similar approach combined with the sparse solver can be used to improve the running time of solving linear programs beyond matrix multiplication for sparse instances when $\omega > 2.5$ (for example, algorithms based on the Strassen algorithm with $\omega \approx 2.808$ [14]). Note that such matrix multiplication algorithms are the ones that are used in practice.

**Theorem 2.4.0.1.** [4-IPM for sparse matrices] Let the matrix multiplication exponent $\omega > 2.5$. Given $A \in \mathbb{R}^{n \times d}$ with full column-rank, $b \in \mathbb{R}^d$, $c \in \mathbb{R}^n$, $1 \leq m \leq n^{1/4}$ an error
parameter $0 < \epsilon < 1$, and a linear program $\min_{A^\top x = b, x \geq 0} c^\top x$ with inner radius $r$ and outer radius $R$, there exists an algorithm that finds $\bar{x} \in \mathbb{R}^n$ such that

$$c^\top \bar{x} \leq \min_{A^\top x = b, x \geq 0} c^\top x + \epsilon \cdot \|c\|_\infty R, \quad \text{and} \quad \|A^\top \bar{x} - b\|_2 \leq \epsilon \cdot \|b\|_2,$$

in time $\tilde{O}\left(\left(\text{NNZ}(A) \cdot m^2 \cdot n + \frac{n^\omega}{m^{2.5}} + n^{2.5}\right) \cdot \log^2\left(\frac{k + \|b\|_2}{\epsilon}\right) \cdot \log\left(\frac{n \cdot R}{\epsilon \cdot r}\right)\right)$ with high probability.

For the case of $\text{NNZ}(A) = O(n)$, if we use the Strassen algorithm and $\kappa / \epsilon$ and $R / r$ are polynomials in $n$, then the above result implies a running time of $\tilde{O}(n^{2.756})$. Moreover, for any $\omega > 2.5$ and $A$ with $\text{NNZ}(A) = o(n^{\omega-1})$, there exists an $m$ such that the above running time is smaller than $n^\omega$.

In Section 2.4.1, we discuss some definitions, parameters, the general IPM approach for solving LPs and our choice of initial feasible solutions. We then prove Theorems 2.1.1.4, 2.1.1.5 and 2.4.0.1 in Sections 2.4.2 to 2.4.4, respectively.

### 2.4.1 LP Preliminaries and Initial Feasible Point

We start by defining the central path. The interior point method first finds a feasible solution on the central path and then following the central path to get close to the optimal solution.

**Definition 2.4.1.1.** A point $x \in \mathbb{R}^n_{\geq 0}$ is on the central path if there exist $s \in \mathbb{R}^n_{\geq 0}, t \in \mathbb{R}_{\geq 0}$ such that

$$x \odot s = \bar{t},$$

$$A^\top x = b,$$

$$Ay + s = c.$$

Note that $x$ is an optimal solution if there exists $s \in \mathbb{R}^n_{\geq 0}$ such that $x \odot s = 0$, and the other two constraints are also satisfied.
The first step of solving linear programs using IPMs is to find an initial feasible solution on the central path. This is achieved by modifying the linear program so that a feasible solution of the modified program is known.

**Definition 2.4.1.2 (Modified linear program).** Consider a linear program

\[ \min_{x: \mathbf{A}^\top x = b, x \geq 0} \mathbf{c}^\top x, \]

with inner radius \( r \) and outer radius \( R \). For any \( \bar{R} \geq 10R, t \geq 8 \|\mathbf{c}\|_{\infty} \bar{R} \), we define the modified primal linear program by

\[ \min_{(\mathbf{x}^+, \mathbf{x}^-, \mathbf{x}^\theta) \in \mathcal{P}_{\bar{R}, t}} \mathbf{c}^\top \mathbf{x}^+ + \bar{\mathbf{c}}^\top \mathbf{x}^-, \]

where

\[ \mathcal{P}_{\bar{R}, t} = \{(\mathbf{x}^+, \mathbf{x}^-, \mathbf{x}^\theta) \in \mathbb{R}^{2n+1}_{\geq 0} : \mathbf{A}^\top (\mathbf{x}^+ - \mathbf{x}^-) = \mathbf{b}, \sum_{i=1}^n x_i^+ + x_i^\theta = \bar{\mathbf{b}}\}, \]

with \( x_i^c = \frac{t}{\mathbf{c}^\top \mathbf{x}^-}, x_i^c = x_i^c - \mathbf{A}(\mathbf{A}^\top \mathbf{A})^{-1} \mathbf{b}, \bar{\mathbf{c}} = t/x_i^c, \bar{\mathbf{b}} = \sum_{i=1}^n x_i^c + \bar{R} \). We define the corresponding dual polytope by

\[ \mathcal{D}_{\bar{R}, t} = \{(\mathbf{s}^+, \mathbf{s}^-, \mathbf{s}^\theta) \in \mathbb{R}^{2n+1}_{\geq 0} : \mathbf{A}y + \lambda \mathbf{1} + \mathbf{s}^+ = \mathbf{c}, -\mathbf{A}y + \mathbf{s}^- = \bar{\mathbf{c}}, \lambda + \mathbf{s}^\theta = 0 \text{ for some } \mathbf{y} \in \mathbb{R}^d, \lambda \in \mathbb{R}\}. \]

Note that defining

\[ \overline{\mathbf{A}} = \begin{bmatrix} \mathbf{A} & 1 \\ -\mathbf{A} & 0 \\ 0 & 1 \end{bmatrix}, \quad \overline{\mathbf{b}} = \begin{bmatrix} \mathbf{b} \\ \mathbf{b} \end{bmatrix}, \quad \text{and} \quad \overline{\mathbf{c}} = \begin{bmatrix} \mathbf{c} \\ \mathbf{c} \\ 0 \end{bmatrix}, \]

the modified primal problem is \( \min_{\mathbf{x}: \overline{\mathbf{A}}^\top \mathbf{x} = \overline{\mathbf{b}}, \mathbf{x} \geq 0} \overline{\mathbf{c}}^\top \mathbf{x} \).

The next lemma states that an initial feasible solution of the modified linear program is known. Moreover starting from that feasible solution, if we decrease the centrality (entries
of the vector $x \odot s$) by an appropriate amount, we can reach a point close to the central path of the original linear program.

**Lemma 2.4.1.3** (Theorem 11 on [100]). Given a linear program $\min_{x:Ax=b, x \geq 0} c^T x$, with inner radius $r$, and outer radius $R$. For any $0 \leq \varepsilon \leq 0.5$, the modified linear program (Definition 2.4.1.2), with $\overline{R} = \frac{5}{6} R$ and $t = 2^{16} \varepsilon^{-3} n^2 R \cdot \|c\|_\infty$, $R$ has the following properties:

1. The point $(x^e_+, x^e_-, \overline{R})$ (as defined in Definition 2.4.1.2) is on the central path of the modified linear program with $(s^+, s^-, s^\theta)$ and $t$, where $s^+ = \overline{t}/x^+$, $s^- = \overline{t}/s^-$, and $s^\theta = t/x^\theta$.

2. For any feasible primal point $\overline{x} = (x^+, x^-, x^\theta) \in P_{\overline{R}, t}$ and dual $\overline{s} = (s^+, s^-, s^\theta) \in D_{\overline{R}, t}$ such that $\frac{5}{6} \|c\|_\infty R \leq x_i s_i \leq \frac{7}{6} \|c\|_\infty R$, we have that $(x^+ - x^-, s^+ - s^\theta) \in P \times D$. In addition, $x^+_i \leq \varepsilon x^+_i$ and $s^\theta_i \leq \varepsilon s^+_i$ for all $i \in [n]$.

As we will show for all of our interior point methods, we can take steps of the form $(1 - \frac{1}{C\sqrt{n}})t$ for some constant $C$. Therefore starting from the initial feasible solution of the modified linear program, we can reach a point close to a feasible solution of the original linear program in $O(\sqrt{n} \log(nR/\varepsilon))$ iterations. We then can run our interior point algorithms on that point to reach a point that is $\varepsilon$ close to the optimal. This can be performed in $O(\sqrt{n} \log(n\|c\|_\infty R/\varepsilon))$ additional iterations. This is illustrated in Algorithm 3. In this algorithm, we denote the IPM algorithms by GENERICIPM since we use different IPMs in Sections 2.4.2 to 2.4.4. Essentially the differences between IPMs is the way they measure the closeness to the central path, the linear systems they solve in each iteration (which is characterized by approximations of the gradient vector and vectors $x$ and $s$ that are used), and the way these linear systems are solved. The former is formalized in the following definition.

**Definition 2.4.1.4.** We consider an algorithm GENERICIPM$(A, x^{(0)}, s^{(0)}, t^{(0)}, t^{(end)}, \varepsilon)$, a generic interior point method, if for a potential function $f$, a function $g$ depending on $n$, and given $x^{(0)}, s^{(0)} \in \mathbb{R}^n, t \in \mathbb{R}$, such that $f(x^{(0)}, s^{(0)}, t^{(0)}) \leq g(n)$, it returns $x^{(end)}, s^{(end)}$
such that $f(x^{(end)}, s^{(end)}, t^{(end)}) \leq g(n)$, and $\|A^\top (x^{(end)} - x^{(0)})\| \leq \varepsilon$. For IPMs based on the 2-norm, $f(x, s, t) = \|(x \odot s - \bar{t})/t\|_2$, and $g(n) = 0.01$. For robust IPMs, $f(x, s, t) = \Phi_{\lambda}((x \odot s - \bar{t})/t)$, where $\Phi_{\lambda}(u) = \sum_{i=1}^n \cosh(\lambda u_i)$, $\lambda > 0$ is a parameter, and $g(n) = 16n$.

An IPM algorithm updates primal and slack vectors $x$ and $s$, in each iteration, by solving the following linear system and setting $x = x + \delta_x$ and $s = s + \delta_s$,

$$
\bar{X}\tilde{\delta}_s + \bar{S}\tilde{\delta}_x = \tilde{\delta}_\mu,
$$
$$
A^\top \tilde{\delta}_x = 0,
$$
$$
A\tilde{\delta}_y + \bar{\delta}_s = 0,
$$

where $\bar{x}, \bar{s}, \bar{\delta}_\mu$ are vectors close (in some norm) to $x, s, \delta_\mu$, and $\delta_\mu$ is a vector function of the gradient of the potential function $f$. Note that we solve these linear systems approximately, but because the error is additive (see Corollary 2.2.0.5), the total feasibility error of the algorithm can be bounded.

We use a robust IPM in Section 2.4.2, and IPMs based on the 2-norm in Section 2.4.3 and Section 2.4.4. In Section 2.4.2, the linear systems are solved by multiplication with an inverse initially obtained by divide-and-conquer algorithms and fast matrix multiplication [17, 105], and maintained by the Woodbury identity under low-rank updates — see Corollary 2.1.4.6. In Section 2.4.3, the linear systems are solved using shifted-number representation [18] — see Theorem 2.4.3.1. In Section 2.4.4, the linear systems are solved by multiplication by representation of inverses obtained by block Krylov method [107, 108, 45, 46], and maintained by the Woodbury identity under low-rank updates. The running time of linear system solvers in Section 2.4.2 and Section 2.4.4 depend on the condition number of the corresponding matrix. Since the modified linear program changes the matrix, we need to argue that its condition number does not blow up compared to the original matrix.
Algorithm 3: Path following interior point method (IPM)

1. **Assumption:** The linear program has inner radius \( r \) and outer radius \( R \).
2. **Input:** Full column rank matrix \( A \in \mathbb{R}^{n \times d} \) and vectors \( b \in \mathbb{R}^d \), \( c \in \mathbb{R}^n \); Error parameters \( 0 < \epsilon_1, \epsilon_2 < 1 \).
3. **Output:** \( \tilde{x} \in Q_{\geq 0}^n \) such that \( \| A^T \tilde{x} - b \|_2 \leq \epsilon_2 \| b \|_2 \) and \( c^T \tilde{x} \leq \min_{x: A^T x = b, x \geq 0} c^T x + \epsilon_1 \).
4. Let \( \epsilon = 1 / (100 \sqrt{n}) \), \( R = 5 \epsilon R \), \( t = 2^{16} \epsilon^{-3} n^2 R / r \), \( \| c \|_\infty R \).
5. Let \( \tilde{A}, \tilde{b}, \tilde{c}, x^+, x^- \) be as defined in Definition 2.4.1.2 for the modified linear programming problem.
6. Let \( x^{(0)} = (x^+, x^-, x^\theta) \) and \( s^{(0)} = t / \tilde{x}^{(0)} \).
7. Let \( (\bar{x}^{(end)}, \bar{s}^{(end)}) = \text{GENERICIPM}(\tilde{A}, \tilde{x}^{(0)}, \tilde{s}^{(0)}, t, \| c \|_\infty R, \epsilon_2) \).
8. Set \( x^{(0)} = \bar{x}^{(end)} \) and \( s^{(0)} = \bar{s}^{(end)} \).
9. Let \( (\bar{x}^{(end)}, \bar{s}^{(end)}) = \text{GENERICIPM}(A, x^{(0)}, s^{(0)}, \| c \|_\infty R, \epsilon / 2n, \epsilon_2) \).

**Lemma 2.4.1.5.** Condition number of \( \tilde{A} \) (as defined in Definition 2.4.1.2) is less than \( 8 \cdot (\kappa(A) + \log(n))^7 \).

**Proof.** First note that the condition number of \( A \) and \( B := \begin{bmatrix} A^T & -A^T & \tilde{0} \end{bmatrix} \) are the same. Therefore setting \( g = \begin{bmatrix} 1^T & 0^T & 1 \end{bmatrix} \), since \( \| g \|_2 \leq \sqrt{n+1} \), and

\[
\| (I - B(B^T B)^{-1} B^T) g \|_2 \geq 1,
\]

by Lemma 2.5.2.7, the condition number of \( \tilde{A} \) is less than \( 8 \cdot (\kappa(A) + \log(n))^7 \). \( \square \)

For the IPMs that use inverse maintenance (Section 2.4.2 and Section 2.4.4), the bit complexities are analyzed in interaction with the inverse, and any rounding required to prevent the bit complexity of the resulting vectors from growing is done when we apply the inverse to a vector. For the IPM that works with shifted numbers to solve the linear systems (Section 2.4.3), given an integer matrix and vector, the exact solution to the linear system is returned as a rational vector. To be sure that the bit complexities (of rational or real vectors) do not blow up, we need to switch between rational and real (fixed-point) vectors. For this purpose, we define the following functions that can be computed in \( \tilde{O}(n\ell) \), where \( \ell \) is the bit complexity of the input vector and \( q \) or \( 1/\epsilon \).
Definition 2.4.1.6. For a vector \( x \in \mathbb{R}^n \), and a number \( q \in \mathbb{Q} \), we define \( \text{QRound}(x, q) \) to be a vector \( u \in \mathbb{Q}^n \), where \( u_i \) is the closest power of \( q \) (or the negative of a power of \( q \)) to \( x_i \). For a vector \( x \in \mathbb{R}^n \) or \( x \in \mathbb{Q}^n \), and a number \( \varepsilon \in \mathbb{R}_{>0} \), we define \( \text{Round}(x, \varepsilon) \) to be a fixed-point vector \( u \in \mathbb{R}^n \), where \( |u_i - x_i| \leq \varepsilon \).

We are now equipped to present our IPMs and analyze their running times in the next sections. Before doing so, we present an example in which the running time of the IPM with shifted numbers (Theorem 2.1.1.5) is better than the IPM based on inverse maintenance (Theorem 2.1.1.4). Let

\[
A = \begin{bmatrix}
1 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\
2 & 1 & 0 & 0 & \cdots & 0 & 0 & 0 \\
0 & 2 & 1 & 0 & \cdots & 0 & 0 & 0 \\
0 & 0 & 2 & 1 & \cdots & 0 & 0 & 0 \\
\vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\
0 & 0 & 0 & 0 & \cdots & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & \cdots & 2 & 1 & 0 \\
0 & 0 & 0 & 0 & \cdots & 0 & 2 & 1 \\
1 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\
\end{bmatrix} \in \mathbb{R}^{n \times (n-1)}, \quad b = \begin{bmatrix} 4/3 \\ 1 \\ 1 \\ 1 \\ \vdots \\ 1 \\ 1 \\ 1 \\ \end{bmatrix} \in \mathbb{R}^{n-1}, \quad c = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \in \mathbb{R}^n.
\]

Then for the following linear program

\[
\min_{x \in \mathbb{R}^{n-1}: A^T x = b, x \geq 0} c^T x,
\]

\( R \leq 2\sqrt{n} \) because for \( x \geq 0 \) with \( \|x\|_2 > 2\sqrt{n} \), there exists \( i \in [n] \) such that \( x_i > 4 \). Then one can see if \( i \neq n \), \( (A^T x)_i > 4 \) \( b_i \), and if \( i = n \), then \( (A^T x)_1 > 4 \) \( b_1 \). Moreover note that for \( x = \frac{1}{3} \cdot c \), we have \( x \geq 0 \) and \( A^T x = b \). Therefore \( r > \frac{1}{3} \). Hence \( R/r \leq \frac{\sqrt{n}}{3} \). However \( \kappa(A) \) as discussed in Section 2.1.3 is at least \( 2^{n-2} \) (check the vectors \( \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \end{bmatrix} \) and \( \begin{bmatrix} (-1/2)^{n-1} & (-1/2)^{n-2} & \cdots & -1/2 & 1 \end{bmatrix} \)) in \( \mathbb{R}^{n-1} \) for the largest and smallest singular value.
respectively). In this case the running time of Theorem 2.1.1.5 is \( \tilde{O}(n^{\omega+0.5} \log^2 (1/ \epsilon)) \) and the running time of Theorem 2.1.1.4 is \( \tilde{O}(n^{\omega+1} \log (1/ \epsilon)) \).

2.4.2 Robust Interior Point Method For Solving Linear Programs

The main result of this section is the following theorem that is achieved by Algorithm 6.

**Theorem 2.1.1.4** (Robust IPM). Given \( A \in \mathbb{R}^{n \times d} \) with full column-rank and condition number \( \kappa \), \( b \in \mathbb{R}^d \), \( c \in \mathbb{R}^n \) all with bit complexity of \( \log(\kappa) \), and an error parameter \( 0 < \epsilon < 1 \), suppose the inner radius and outer radius of the linear program \( \min_{Ax=b, x \geq 0} c^T x \) is \( r \) and \( R \), respectively. Then there is an algorithm that computes \( \tilde{x} \in \mathbb{R}^n \) such that

\[
c^T \tilde{x} \leq \min_{A^T x = b, x \geq 0} c^T x + \epsilon \quad \text{and} \quad \|A^T \tilde{x} - b\|_2 \leq \epsilon,
\]

in time \( \tilde{O}\left(\left(n^{\omega} + n^{2.5-\alpha/2} + n^{2+1/6}\right) \cdot \log\left(\frac{\kappa R}{\epsilon F}\right) \cdot \log\left(\frac{R}{\epsilon r}\right)\right) \).

For this result we work with the potential function of the form \( \Phi_1(\frac{x s}{r} - 1) \), where

\[
\Phi_1(u) = \sum_{i=1}^{n} \cosh(\lambda u_i) = \sum_{i=1}^{n} \frac{\exp(\lambda u_i) + \exp(-\lambda u_i)}{2}.
\]

We use the following data structure to maintain the projection matrix \( A (A^T \overline{X} S^{-1}) A^T \) and compute the changes \( \overline{\delta}_s \) and \( \overline{\delta}_x \).

**Theorem 2.4.2.1** (Projection maintenance data structure — PDS). Let \( n \geq d \). There exists a data structure with the following operations that maintain an explicit matrix \( Z^{-1} \).

- **INITIALIZE**\((A \in \mathbb{R}^{n \times d}, x \in \mathbb{R}^n, s \in \mathbb{R}^n, r \in \mathbb{R}^n, f : \mathbb{R} \to \mathbb{R}, \alpha, \epsilon)\): Sets \( \overline{x} = x, \overline{s} = s, \overline{r} = r \), and computes a linear operation \( \overline{Z} \) such that

\[
\|\overline{Z} - A (A^T \overline{X} S^{-1}) A^T\|_F \leq \kappa^2 \cdot \epsilon.
\]

Moreover, sets \( \overline{r} = \overline{x} = \overline{s} = \overline{0} \in \mathbb{R}^n \), \( T = \emptyset \), and sets \( \overline{w} = \overline{Z} S^{-1} f(\overline{r}) \).
• \textbf{UPDATE}(S \subseteq \{1, \ldots, n\}, x^{(u)} \in \mathbb{R}^{|S|}, s^{(u)} \in \mathbb{R}^{|S|}, r^{(u)} \in \mathbb{R}^{|S|}): Sets \overline{x}_S = x^{(u)} - \overline{x}, \overline{s}_S = s^{(u)} - \overline{s}, \overline{r}_S = r^{(u)} - \overline{r}, and T = T \cup S. If |T| > n^{\tilde{a}}, sets \overline{x} = \overline{x} + \overline{s}, \overline{s} = \overline{s} + \overline{s}, \overline{r} = \overline{r} + \overline{r}, T = \emptyset, and updates \tilde{Z} such that after k calls to \textbf{UPDATE}, we have

\[
\| \tilde{Z} - A(A^T \overline{X}^{-1} A)A^T \|_F \leq 50\kappa^{12} \cdot (512 \cdot k \cdot (5\kappa^5)^{26} + 2 \cdot k \cdot (6\kappa^5)^2 + 1) \cdot \varepsilon,
\]

where \(\kappa > n + d\) is a number such that

\[
\| A \|_F, \| X^{-1} \|_F, \| X^{-1} S \|_F, \| X^{-1} S^{-1} \|_F, \| X^{-1} S^{-1} \|_F, \\
\| A^T X S^{-1} A \|_F, \| (A^T X S^{-1} A)^{-1} \|_F \leq \kappa,
\]

during all the updates and

\[
\varepsilon < \frac{1}{10\kappa^5 \cdot (512 \cdot (k + 1) \cdot (5\kappa^5)^{26} + 2 \cdot k \cdot (6\kappa^5)^2 + 1)}.
\]

After updating \(\tilde{Z}\), it sets \(\tilde{w} = \tilde{Z}\overline{S}^{-1} f(\overline{r})\).

• \textbf{QUERY}(): \textbf{Compute} \(D^{-1} \in \mathbb{R}^{|T| \times |T|}\) \textbf{such that}

\[
\| D^{-1} - \left( \tilde{X}_{T,T}^{-1} \tilde{S}_{T,T} + \tilde{Z}_{T,T} \right)^{-1} \|_F \leq \varepsilon.
\]

Then it computes and returns

\[
\tilde{w} + \tilde{Z}(\overline{S}^{-1})_T^T f(\overline{r}) - \tilde{Z}_{T,T} D^{-1}(\tilde{Z}_{T,T})^T \overline{S}^{-1} f(\overline{r} + \overline{r}).
\]

The running time of initialize, update, and query are \(n^\omega \cdot \tilde{O}(\log(k/\varepsilon))\), \(\text{MM}(n, n, |T|) \cdot \tilde{O}(\log(k/\varepsilon))\), and \((n^{1+\tilde{a}} + n^{\tilde{a} \omega}) \cdot \tilde{O}(\log(k/\varepsilon))\), respectively.

\textbf{Proof.} We show that the data structure in Algorithms 4 and 5 achieves the desired result.
First note that $M$ is invertible since $SX^{-1}$, and $A^T XS^{-1} A$ are invertible and

$$M^{-1} = \begin{bmatrix} (A^T XS^{-1} A)^{-1} & -(A^T XS^{-1} A)^{-1} A^T XS^{-1} \\ -XS^{-1} A (A^T XS^{-1} A)^{-1} & XS^{-1} + XS^{-1} A (A^T XS^{-1} A)^{-1} A^T XS^{-1} \end{bmatrix}. $$

Note that by triangle inequality $\|M\|_F \leq 3\kappa$, and $\|M^{-1}\|_F \leq 2\kappa + 2\kappa^3 + \kappa^5$. Since $Z$ is a matrix with $\|Z^{-1} - M\|_F \leq \epsilon$, taking $\tilde{Z}$ to be the $n$-by-$n$ bottom right block of $Z$, then

$$\tilde{Z} = S X^{-1} (Z - XS^{-1}) X^{-1} S$$

is a linear operator for $A (A^T XS^{-1} A)^{-1} A^T$.

We now bound the running times. The initialization requires computing the inverse of an $(n + d) \times (n + d)$ matrix with $n \geq d$. The required error bound and the condition number bounds give a running time of $n^\omega \cdot \tilde{O}(\log(\kappa/\epsilon))$. Then computing $\tilde{Z}$ and $\tilde{w}$ according to line 7 and line 8 of Algorithm 4 is done in $\tilde{O}(n^2 \log(\kappa/\epsilon))$ time since $X$ and $S$ are diagonal.

We now bound the running time of the update. If $|T| \leq n^{\tilde{\alpha}}$, then the cost is bounded by $\tilde{O}(n \log(\kappa/\epsilon))$ since we only set new values for entries of vectors according to the input. Otherwise, we update the inverse. Computing $D^{-1}$ takes $|T|^\omega \cdot \tilde{O}(\log(\kappa/\epsilon))$ because of the error bound and condition number bounds and the fact that $U Z U$ only selects a submatrix of $Z$. Then computing $Z U^T D^{-1} U Z$ takes $M(n, n, |T|) \cdot \tilde{O}(\log(\kappa/\epsilon))$ and having this matrix, we can update $Z$ in time $\tilde{O}(n^2 \log(\kappa/\epsilon))$. Finally, for the update, we need to recompute $\tilde{w}$, which can be done in $\tilde{O}(n^2 \log(\kappa/\epsilon))$ time, similar to the initialization step.

Since the update procedure ensures that $|T| \leq n^{\tilde{\alpha}}$, computing $D^{-1}$ in the query procedure takes at most $n^\omega \cdot \tilde{O}(\log(\kappa/\epsilon))$. Then computing

$$\tilde{w} + \tilde{Z} (S^{-1})_T^T f(\tilde{r}_T) - \tilde{Z}_T D^{-1} (\tilde{Z}_T)^T S^{-1} f(\tilde{r} + \tilde{r})$$

according to the query step of Algorithm 5 takes only $n^{1+\tilde{\alpha}} \cdot \tilde{O}(\log(\kappa/\epsilon))$ time. Note that we do not form the matrix $\tilde{Z}$ for this procedure because forming this matrix would impose a cost of $\Omega(n^2)$.

We now need to bound the error of our matrix after $k$ updates. We prove this by induc-
tion. For the base case, note that $\|Z^{-1} - M\|_F \leq \varepsilon$. Therefore since $\|M\|_F, \|M^{-1}\|_F \leq 5\kappa^5$, we have

$$\left\| Z - (\bar{X}^{-1} + \bar{X}^{-1} A (A^T \bar{X}^{-1} A)^{-1} A^T \bar{X}^{-1}) \right\|_F \leq 5\kappa^5 \cdot \varepsilon.$$ 

Therefore

$$\left\| \tilde{Z} - A (A^T \bar{X}^{-1} A)^{-1} A^T \right\|_F \leq \left\| \bar{X}^{-1} S (\tilde{Z} - \bar{X}^{-1}) \bar{X}^{-1} S - A (A^T \bar{X}^{-1} A)^{-1} A^T \right\|_F \leq 5\kappa^7 \cdot \varepsilon.$$ 

Now suppose after $k$ updates

$$\|Z^{-1} - M\|_F \leq (512 \cdot k \cdot (5\kappa^5)^{26} + 2 \cdot k \cdot (6\kappa^5)^2 + 1) \cdot \varepsilon,$$

and $\bar{X}^{-1}$ is the diagonal matrix corresponding to the $k + 1$'st update. Let

$$Y = Z - ZU^T D^{-1} UZ , \quad \left\| D^{-1} - \left( \bar{X}^{-1} S \bar{X}^{-1} + UZU^T \right)^{-1} \right\|_F \leq \varepsilon,$$

and $\tilde{Y}$ is obtained by rounding the entries of $Y$ to numbers with $\lceil \log(100(n + d)^2 \cdot \kappa^5 / \varepsilon) \rceil$ bits. Lemma 2.1.2.2 directly gives

$$\left\| Y^{-1} - \tilde{M} \right\|_F \leq (512 \cdot (k + 1) \cdot (5\kappa^5)^{26} + 2 \cdot k \cdot (6\kappa^5)^2 + 1) \cdot \varepsilon,$$

where $\tilde{M}$ is the matrix $M$ after the update, i.e.,

$$\tilde{M} = M + \begin{bmatrix} 0_{d \times d} & 0_{d \times n} \\ 0_{n \times d} & \bar{S}^{-1} \end{bmatrix}.$$
Moreover the norm bound assumptions imply $\|\tilde{M}\|_F, \|\tilde{M}^{-1}\|_F \leq 5\kappa^5$, and
\[
\|Y^{-1} - \tilde{M}\|_F \leq (512 \cdot (k + 1) \cdot (5\kappa^5)^{26} + 2 \cdot k \cdot (6\kappa^5)^2 + 1) \cdot \varepsilon < \frac{1}{10\kappa^5},
\]
by Lemma 2.3.0.1, we have $\|Y\|_F \leq 10\kappa^5$. Therefore the number of bits before decimal for the entries $Y$ is bounded by $\log_2(10\kappa^5) + 1$. Therefore the rounding only introduces error in the bits after the decimal point. Therefore
\[
\|Y - \tilde{Y}\| \leq \varepsilon.
\]
Invoking Lemma 2.3.0.1 and because $\varepsilon < \frac{1}{10\kappa^5}$ (by assumption), $\|Y\|_F \leq 10\kappa^5$, and
\[
\|Y^{-1}\|_F \leq \|Y^{-1} - \tilde{M}\|_F + \|\tilde{M}\|_F \leq \frac{1}{10\kappa^5} + 5\kappa^5 \leq 6\kappa^5,
\]
we have $\|\tilde{Y}^{-1} - Y^{-1}\|_F \leq 2 \cdot (6\kappa^5)^2 \cdot \varepsilon$. Therefore by triangle inequality,
\[
\|\tilde{Y}^{-1} - \tilde{M}\|_F \leq \|\tilde{Y}^{-1} - Y^{-1}\|_F + \|Y^{-1} - \tilde{M}\|_F
\]
\[
\leq (512 \cdot (k + 1) \cdot (5\kappa^5)^{26} + 2 \cdot (k + 1) \cdot (6\kappa^5)^2 + 1) \cdot \varepsilon.
\]
Therefore
\[
\|\tilde{Y} - \tilde{M}^{-1}\|_F \leq 2 \cdot (5\kappa^5)^2 \cdot (512 \cdot (k + 1) \cdot (5\kappa^5)^{26} + 2 \cdot (k + 1) \cdot (6\kappa^5)^2 + 1) \cdot \varepsilon.
\]
Therefore after the update, we have,
\[
\|\tilde{Z} - A(A^\top X S^{-1} A)A^\top\|_F \leq 2\kappa^2 \cdot (5\kappa^5)^2 \cdot (512 \cdot (k + 1) \cdot (5\kappa^5)^{26} + 2 \cdot (k + 1) \cdot (6\kappa^5)^2 + 1) \cdot \varepsilon.
\]
Lemma 2.4.2.2 ([100]). Let \( \mathbf{x}, \mathbf{s} \) be vectors with \( \| \log \mathbf{x} - \log \mathbf{x} \|_\infty \leq \frac{1}{4\delta}, \| \log \mathbf{s} - \log \|_\infty \leq
Algorithm 5: Projection maintenance data structure (PDS) - Part 2

1 Variables: \( n, d \in \mathbb{N} \) with \( n > d \), \( A \in \mathbb{R}^{n \times d} \), \( Z^{-1} \in \mathbb{R}^{d \times d} \), \( \bar{x}, \bar{s}, \bar{r}, \bar{x}, \bar{s}, \bar{r} \in \mathbb{R}^{n} \),
\( 1 > \varepsilon > 0 \).

2 Procedure QUERY()

3 Let \( \bar{Z} \) be the \( n \times n \) bottom right block of \( Z \)

4 Compute \( D^{-1} \in \mathbb{R}^{|T| \times |T|} \) such that
\[ \| D^{-1} - (\bar{X}_{T,T} \bar{S}_{T,T} + (S\bar{X})^{-1})_{T,T}(\bar{Z}_{T,T} - (X\bar{S})^{-1}T, T)\|_F \leq \varepsilon \]

5 Compute
\[
\begin{align*}
\mathbf{h}^{(1)} &= S\bar{X}^{-1} \left( (\bar{Z}(\bar{X}^{-1}_{T,T})^\top f(\bar{r}_T) - (\bar{S}^{-1}_{T,T})^\top f(\bar{r}_T) \right), \text{ and} \\
\mathbf{h}^{(2)} &= D^{-1} \left( (S\bar{X}^{-1})_{T,T} \left( \bar{Z}_T \bar{X}^{-1} f(\bar{r} + \bar{r}) - (\bar{S}^{-1})_{T,T} f(\bar{r} + \bar{r}) \right) \right), \text{ and} \\
\mathbf{h}^{(3)} &= S\bar{X}^{-1} \left( (\bar{Z}_T^\top) (S\bar{X}^{-1})_{T,T} \mathbf{h}^{(2)} - (I_T)^\top \mathbf{h}^{(2)} \right)
\end{align*}
\]

6 Compute and return \( \bar{w} + \mathbf{h}^{(1)} - \mathbf{h}^{(3)} \)

\[ \frac{1}{48}, \ W = \bar{X}S^{-1}, \text{ and } P = \sqrt{W}(A^\top WA)^{-1} \sqrt{W}. \text{ Moreover let} \]
\[ \delta_x := \frac{\bar{X}}{\sqrt{xs}} (I - P) \frac{I}{\sqrt{xs}} \bar{\delta}_\mu, \text{ and } \delta_s := \frac{\bar{S}}{\sqrt{xs}} P \frac{I}{\sqrt{xs}} \bar{\delta}_\mu, \]

where \( \bar{\delta}_\mu = -\frac{t}{2\lambda_1 \| \Phi_\lambda(\bar{r}) \|_2} \), \( \bar{r} \) is a vector with \( \| \bar{r} - \frac{xs - t}{t} \|_\infty \leq \frac{1}{48\lambda} \), and \( \lambda = 16 \log 40n. \) Then under the invariant \( \Phi_\lambda(\frac{xs - t}{t}) \leq 16n, \)
\[ \left\| \frac{x \otimes s - t}{t} \right\|_\infty \leq \frac{1}{16}, \text{ and } \| \delta_x/x \|_2 \leq \frac{1}{20\lambda}, \text{ and } \| \delta_s/s \|_2 \leq \frac{1}{20\lambda}. \]

The bound \( \| \frac{xs - t}{t} \|_\infty \leq \frac{1}{16} \) on the above lemma directly follows from \( \Phi_\lambda(\frac{xs - t}{t}) \leq 16n \) and does not depend on the computation of \( \delta_x \) and \( \delta_s \).

Remark 2.4.2.3. The upper bounds stated in [100] for \( \| \delta_x/x \|_2 \) and \( \| \delta_s/s \|_2 \) is \( 1/(16\lambda) \), but it can easily be strengthened to the bounds we stated above with the same argument.

Now note that instead of \( \delta_x \) and \( \delta_s \) we compute \( \bar{\delta}_x \) and \( \bar{\delta}_s \) by Corollary 2.2.0.5. Note
that we use the matrix itself as the preconditioner and therefore, we only take one step
by Corollary 2.2.0.5. This gives \( \tilde{u} \) such that \( \| \tilde{u} - u \|_2 \leq \epsilon \| u \|_2 \), where \( u = P \frac{1}{\sqrt{XS}} \delta \mu \).

Therefore

\[
\left\| \tilde{x} \right\|_2 \leq \frac{1}{20.1} + \left\| \delta_x - \delta_x \right\| \leq \frac{1}{20.1} + \frac{I}{\sqrt{XS}} (\tilde{u} - u) \leq \frac{1}{20.1} + \frac{\epsilon}{\min_{i \in [n]} \sqrt{\tilde{x}_i s_i}} \| u \|_2
\]

\[
\leq \frac{1}{20.1} + \frac{\epsilon \cdot \max_{i \in [n]} \sqrt{\tilde{x}_i s_i}}{\min_{i \in [n]} \sqrt{\tilde{x}_i s_i}} \frac{I}{\sqrt{XS}} u \leq \frac{1}{20.1} + \frac{\epsilon}{\min_{i \in [n]} \sqrt{\tilde{x}_i s_i}} \| u \|_2
\]

\[
\leq \frac{1}{20.1} + \frac{\epsilon \cdot \max_{i \in [n]} \sqrt{\tilde{x}_i s_i}}{20.1 \cdot \min_{i \in [n]} \sqrt{\tilde{x}_i s_i}} \| \delta_x \|_2
\]
A similar argument gives the same bound for \( \|\tilde{s}/\tilde{s}\|_2 \). Note that since entries of \( x \odot s \) are close to \( t \) and \( \tilde{x} \) and \( \tilde{s} \) are close to \( x \) and \( s \), respectively, we can take \( \varepsilon = \Omega(t^{(0)}/t^{(end)}) \), so that \( \|\tilde{s}/\tilde{s}\|_2 \leq 1/16\varepsilon \) and \( \|\tilde{x}/\tilde{x}\|_2 \leq 1/16\varepsilon \).

**Remark 2.4.2.4.** Note that although we compute a vector \( \tilde{u} \) using Corollary 2.2.0.5 and use that to compute \( \tilde{s} \) instead of using \( u = P(1/\sqrt{x}s) \tilde{\mu} \) to compute them, we still have \( \tilde{S}\tilde{s} + \tilde{x}\tilde{s} = \tilde{\mu} \) because

\[
\tilde{S}\tilde{s} + \tilde{x}\tilde{s} = \tilde{\mu} - \sqrt{X}S\tilde{u} + \sqrt{X}S\tilde{u} = \tilde{\mu}
\]

The following is a combination of Lemma 16 and 18 of [100] that essentially follows from Remarks 2.4.2.3 and 2.4.2.4 by the same proof.

**Lemma 2.4.2.5 ([100]).** Let \( \lambda = 16 \log 40n \), \( t^{(0)} \in \mathbb{R}_{>0} \), and \( x^{(0)}, s^{(0)} \in \mathbb{R}^n \) such that \( \Phi_\lambda(x^{(0)} \odot s^{(0)}/t^{(0)}) \leq 16n \). Moreover for \( k \in \mathbb{N} \), let \( x^{(k)} = x^{(k-1)} + \tilde{x}^{(k)} \) and \( s^{(k)} = s^{(k-1)} + \tilde{s}^{(k)} \) be computed by an iteration of robust IPM (Algorithm 6) such that \( \|\tilde{x}^{(k)}/\tilde{s}^{(k-1)}\|_2 \leq 1/16\varepsilon \) and \( \|\tilde{s}^{(k)}/\tilde{x}^{(k)}\|_2 \leq 1/16\varepsilon \) where \( \tilde{x}^{(k)}, \tilde{s}^{(k)} \) are vectors that satisfy \( \|\log \tilde{x}^{(k)} - \log x^{(k)}\|_\infty \leq 1/48 \), \( \|\log \tilde{s}^{(k)} - \log s^{(k)}\|_\infty \leq 1/48 \). Then for \( r^{(k)} := x^{(0)} \odot s^{(0)}/t^{(0)} \), \( \|r^{(k+1)} - r^{(k)}\|_2 \leq 1/16\varepsilon \). Moreover \( \|\log x^{(k+1)} - \log x^{(k)}\|_2, \|\log s^{(k+1)} - \log s^{(k)}\|_2 \leq 1/8\varepsilon \). In addition \( \Phi_\lambda(r^{(k+1)}) \leq 8n \) if \( \Phi_\lambda(r^{(k)}) \leq 8n \), and \( \Phi_\lambda(r^{(k+1)}) \leq \Phi_\lambda(r^{(k)}) \), otherwise.

The next lemma is useful for bounding the running time of inverse maintenance in Algorithm 6.

**Lemma 2.4.2.6 ([100]).** Let \( \beta > 0 \) be a constant. Let \( v^{(0)}, v^{(1)}, v^{(2)}, \ldots \) be vectors in \( \mathbb{R}^n \) arriving in a stream with the guarantee that \( \|v^{(k+1)} - v^{(k)}\|_2 \leq \beta \) for all \( k \). Then for \( 0 < C < 0.5 \), we can pick \( \tilde{v}^{(0)}, \tilde{v}^{(1)}, \tilde{v}^{(2)}, \ldots \), so that (see Algorithm 4 on [100])

- \( \|\tilde{v}^{(k)} - v^{(k)}\|_\infty \leq C \) for all \( k \).
- \( \|\tilde{v}^{(k)} - \tilde{v}^{(k-1)}\|_0 \leq O(2^{2q_k}(\beta/C)^2 \log^2(n)) \) where \( q_k \) is the largest integer with \( k = 0 \mod 2^{q_k} \).
We are now equipped to prove our main result regarding the bit complexity of solving LPs.

**Proof of Theorem 2.1.1.4.** We prove that Algorithm 6 converges and outputs a near-feasible solution, and we analyze the running time and bit complexity of this algorithm. Then this is combined with Algorithm 3 and Lemma 2.4.1.3 (to find the initial feasible solution) to give the desired result.

Note that by Lemmas 2.4.2.2 and 2.4.2.5, we have

\[ ||x \odot s - t||_\infty \leq \frac{1}{16}, \]

where \( x, s \) are the output of Algorithm 6 and \( t > 0 \) is a number smaller than \( t^{(end)} \). Therefore

\[ c^T \hat{x} \leq \min_{x : A^T x = b, x \geq 0} c^T x + \frac{n \cdot \hat{t}}{16}. \]

Taking \( t^{(end)} \) to be small enough, we can guarantee an upper bound on the error. We now discuss the feasibility of the returned solution. First, note that \( \hat{x} \geq 0 \) by induction through the guarantee of Remark 2.4.2.3. Moreover in each iteration of Algorithm 6, we compute \( \tilde{\delta}_x \) as

\[ \tilde{\delta}_x = \frac{I}{S} \tilde{\delta}_\mu - \frac{X}{\sqrt{XS}} \tilde{u}, \]

where \( \tilde{u} \) is a vector with \( ||\tilde{u} - u||_2 \leq \varepsilon ||u||_2 \) and \( u = P \frac{1}{\sqrt{XS}} \tilde{\delta}_\mu \). Therefore by Corollary 2.2.0.5, and the bounds on the condition number of \( \tilde{A}/\tilde{S} \) and \( A \),

\[ \left\| A^T \tilde{\delta}_x \right\|_2 \leq \varepsilon \cdot \kappa \cdot \frac{R}{r} \left\| u \right\|_2 \leq \varepsilon \cdot \kappa \cdot \frac{R}{r} \left\| \frac{1}{\sqrt{XS}} \tilde{\delta}_\mu \right\|_2, \]

where the last inequality follows because \( P \) is a projection matrix. Now since \( \tilde{\delta}_\mu = -\frac{t \cdot \nabla \Phi(x)}{32.1 \cdot \|\nabla \Phi(x)\|_2} \) and by Lemma 2.4.2.2 \( \| \tilde{r} - \frac{X \odot s - t}{r} \|_\infty \leq \frac{1}{48T} \) and \( \| \frac{X \odot s - t}{r} \|_\infty \leq \frac{1}{16} \), \( \left\| \frac{1}{\sqrt{XS}} \tilde{\delta}_\mu \right\|_2 \) is bounded by poly\( (n) \). Therefore setting \( \varepsilon = \frac{T \cdot \kappa \cdot \frac{R}{r} \left\| \frac{1}{\sqrt{XS}} \tilde{\delta}_\mu \right\|_2}{T \cdot \kappa \cdot \frac{R}{r} \left\| \frac{1}{\sqrt{XS}} \tilde{\delta}_\mu \right\|_2} \), where \( T \) is the number of iter-
ations of the algorithm, by triangle inequality we have the guarantee that \( \| A^T \tilde{x} - b \|_2 \leq \epsilon \).

We now bound the running time of the algorithm. Based on the errors we discussed above and Corollary 2.2.0.5 and Remark 2.2.0.4, we need to take the bit complexity of \( \tilde{O}(\log(\frac{kR}{\epsilon r})) \) for our inverses. By picking the right constants according to Lemma 2.1.2.2, we can guarantee the stability of inverse maintenance and the data structure used in Algorithm 6. By Theorem 2.4.2.1, the data structure is initialized in time \( \tilde{O}(n^{0.5} \log(\frac{\epsilon}{\epsilon r})) \).

By construction of Algorithms 3 and 6, the number of iterations of our IPM is

\[
\tilde{O}(n^{0.5} \log(\frac{R}{\epsilon \cdot r})).
\]

By Lemmas 2.4.2.5 and 2.4.2.6, for each \( q_k \), there are \( \tilde{O}(\frac{n^{0.5}}{2^{q_k}} \log(\frac{R}{\epsilon \cdot r})) \) iterations where \( \tilde{O}(2^{q_k}) \) entries of our vectors \( \tilde{x}, \tilde{s}, \tilde{r} \) change. Then by Theorem 2.4.2.1, the running time of inverse maintenance is bounded by

\[
\sum_{q_k:2^{q_k} > n^{\tilde{c}}} \tilde{O} \left( \frac{n^{0.5}}{2^{q_k}} \log(\frac{R}{\epsilon \cdot r}) \cdot \text{MM}(n, n, 2^{q_k}) \log(\frac{k \cdot R}{\epsilon \cdot r}) \right),
\]

where \( \tilde{c} = \min\{\alpha, 2/3\} \) and \( \alpha \) is the dual of the matrix multiplication exponent. By Lemma 2.1.4.1, this is

\[
\sum_{q_k:2^{q_k} > n^{\tilde{c}}} \tilde{O} \left( \frac{n^{0.5}}{2^{q_k}} \log(\frac{R}{\epsilon \cdot r}) \cdot \left( n^2 + (2^{q_k})^{2(\omega-2)/(1-\alpha)} n^{2-\alpha(\omega-2)/(1-\alpha)} \right) \log(\frac{k \cdot R}{\epsilon \cdot r}) \right),
\]

Since \( 2^{q_k} > n^{\tilde{c}} \), \( \frac{n^{2.5}}{2^{q_k}} \leq \max\{n^{2.5-\alpha/2}, n^{2+1/6}\} \). Moreover the term

\[
n^{2.5-\alpha(\omega-2)/(1-\alpha)} \cdot (2^{q_k})^{2(\omega-2)/(1-\alpha)-1}
\]

is either increasing or decreasing in \( q_k \) depending on whether \( 2(\omega - 2)/(1 - \alpha) - 1 \) is positive or negative. If \( 2(\omega - 2)/(1 - \alpha) - 1 > 0 \), the maximum happens for \( 2^{q_k} = n^{0.5} \) in
which case

\[ n^{2.5-\alpha \cdot (\omega-2)/(1-\alpha)} \cdot (2^q)^{2(\omega-2)/(1-\alpha)} - 1 = n^\omega. \]

If \(2(\omega - 2)/(1 - \alpha) - 1 \leq 0\), the maximum happens for \(2^q = \tilde{n}^{\alpha/2}\) in which case

\[ n^{2.5-\alpha \cdot (\omega-2)/(1-\alpha)} \cdot (2^q)^{2(\omega-2)/(1-\alpha)} - 1 = n^{2.5-\alpha \cdot (\omega-2)/(1-\alpha)} \cdot \tilde{n}^{\alpha/2}. \]

If \(\tilde{\alpha} = \alpha\), then this is bounded by \(n^{2.5-\alpha/2}\). Otherwise, \(\tilde{\alpha} < \alpha\) and this is bounded by \(n^{2+1/6}\). We finally need to bound the running time of queries to the data structure which by Theorem 2.4.2.1 is

\[ \tilde{O}(n^{0.5} \log(\frac{R}{\epsilon \cdot r})(n^{1+\tilde{\alpha}} + n^{\tilde{\alpha} \omega}) \cdot \log(\frac{\kappa \cdot R}{\epsilon \cdot r})), \]

which since \(\tilde{\alpha} \leq \frac{2}{3}\) is bounded by

\[ \tilde{O}((n^{2+1/6} + n^{0.5+2\omega/3}) \cdot \log(\frac{R}{\epsilon \cdot r}) \cdot \log(\frac{\kappa \cdot R}{\epsilon \cdot r})). \]

Now since \(\omega \geq 2\), \(0.5 \leq \omega/4\) and therefore \(0.5 + 2\omega/3 \leq 11\omega/12 < \omega\). Combining the above running times gives the desired result.

\[ \square \]

2.4.3 Inverse Free Path Following IPM

The main result of this section is the following theorem that is achieved by substituting Algorithm 7 in line 7 and line 9 of Algorithm 3. For this, we use a version of IPM that works with a 2-norm potential function, uses \(\bar{x}\), and \(\bar{s}\) that have a relative error of 0.01 compared to \(x\) and \(s\), and uses a \(\tilde{\delta}_\mu\) that has a relative error of 0.001 compared to \(\delta_\mu\).

Theorem 2.1.1.5 (Inverse-free IPM). Given \(A \in \mathbb{R}^{n \times d}\) with full column-rank, \(b \in \mathbb{R}^d\), \(c \in \mathbb{R}^n\) all with bit complexity of \(\ell\), and an error parameter \(0 < \epsilon < 1\), suppose the inner radius and outer radius of the linear program \(\min_{A^\top x = b, x \succeq 0} c^\top x\) is \(r\) and \(R\), respectively.
Then there is an algorithm that finds $\hat{x} \in \mathbb{R}^n$ such that

$$c^T\hat{x} \leq \min_{A^Tx = b, x \geq 0} c^T x + \epsilon, \text{ and } \|A^T\hat{x} - b\|_2 \leq \epsilon,$$

in time $O(n^{\omega+0.5} \cdot (\ell + \log(\frac{R}{\epsilon})) \cdot \log(\frac{R}{\epsilon}))$.

To solve the linear system corresponding to each iteration of the IPM, we use the following result that solves an integer linear system in the true matrix multiplication time times the bit complexity of the input (without paying $\log(\kappa)$ in the running time).

**Theorem 2.4.3.1** ([18]). Let $M \in \mathbb{Z}^{n \times n}$ be an invertible matrix and $b \in \mathbb{Z}^n$. There exists a Las Vegas algorithm that returns $M^{-1}b \in \mathbb{Q}^n$ with a expected cost of $O(n^\omega \cdot (\log n) \cdot (\log \|M\|_\infty + \frac{\log \|b\|_\infty}{n} + \log n) \cdot C^2)$, where $C = \log((\log \|M\|_\infty + \frac{\log \|b\|_\infty}{n} + \log n))$.

The above theorem uses Cramer’s rule to compute the solution of a linear system.

**Fact 2.4.3.2** (Cramer’s rule). Let $M$ be a nonsingular $n$-by-$n$ matrix, $b$ be an $n$-vector, and $x$ be a solution to the linear system $Mx = b$. Then $x_i = \frac{\det(M_i)}{\det(M)}$, where $M_i$ is the matrix obtained by replacing column $i$ of $M$ with $b$.

Therefore given an integer linear system, the algorithm of Theorem 2.4.3.1 returns the solution of $Mx = b$ as a rational number where the bit complexity of the denominators is $O(n \log \|M\|_\infty)$, and the bit complexity of the numerators is $O(n \log \|M\|_\infty + \log \|b\|_\infty)$.

Note that since the entries of $x$ have a common denominator of $\det(M)$, the vector $\det(M)x$ is an integer vector and its bit complexity (up to constant factors) is the same as the bit complexity of $x$, i.e., $O(n \log \|M\|_\infty + \log \|b\|_\infty)$. Therefore the solution of the $Mx = b$ can be used for the right-hand side of another linear system without increasing the running time solving the linear system — note that the bit complexity of $b$ in Theorem 2.4.3.1 appears with a factor of $1/n$ in the running time.

If the right-hand side of the linear system does not have common denominators, then turning the right hand side to an integer vector can increase the bit complexity by $n$ times.
the bit complexity of the denominators. Thus if the bit complexity of the denominators is \( n \), this leads to a bit complexity of \( n^2 \) which leads to increasing the running time of solving the linear system to \( n^{\omega+1} \). To avoid this, we make sure that the right-hand side of the linear systems we solve have common denominators. This is achieved by rounding the vectors \( \mathbf{x}, \mathbf{s}, \delta_{\mu} \) to rational vectors \( \mathbf{\tilde{x}}, \mathbf{\tilde{s}}, \delta_{\mu} \) close to them.

To guarantee that the right-hand side of linear systems we solve have common denominators, in each iteration, instead of using \( \mathbf{x}, \mathbf{s}, \) and \( \delta_{\mu} \) in the steps of IPM, we use \( \mathbf{\tilde{x}}, \mathbf{\tilde{s}}, \delta_{\mu} \), where entries of \( \mathbf{\tilde{x}}, \mathbf{\tilde{s}} \) are the closest powers of \( \frac{101}{100} \) to entries of \( \mathbf{x}, \mathbf{s} \), and entries of \( \delta_{\mu} \) are the closest powers of \( \frac{1001}{1000} \) to entries of \( \delta_{\mu} \). In this section we show the IPM still converges with these vectors. Therefore, we make updates by solving the following linear system

\[
\begin{align*}
\mathbf{X}\delta_s + \mathbf{S}\delta_x &= \delta_{\mu}, \\
\mathbf{A}^\top\delta_x &= 0, \\
\mathbf{A}\delta_y + \delta_s &= 0.
\end{align*}
\]

Lemma 2.4.3.3. The solution of (Equation 2.4.2) is

\[
\delta_s = \mathbf{A}(\mathbf{A}^\top\mathbf{X}\mathbf{S}^{-1}\mathbf{A})^{-1}\mathbf{A}^\top\mathbf{S}^{-1}\delta_{\mu}, \text{ and } \delta_x = \mathbf{S}^{-1}\delta_{\mu} - \mathbf{X}\mathbf{S}^{-1}\delta_s.
\]

Proof. First note that since \( \mathbf{X}\delta_s + \mathbf{S}\delta_x = \delta_{\mu}, \mathbf{S}\delta_x = \delta_{\mu} - \mathbf{X}\delta_s \). Therefore multiplying \( \mathbf{S}^{-1} \) from left and using the fact that diagonal matrices commute with each other,

\[
\delta_x = \mathbf{S}^{-1}\delta_{\mu} - \mathbf{X}\mathbf{S}^{-1}\delta_s.
\]

Therefore since \( \mathbf{A}^\top\delta_x = 0, \)

\[
\mathbf{A}^\top\mathbf{X}\mathbf{S}^{-1}\delta_s = \mathbf{A}^\top\mathbf{S}^{-1}\delta_{\mu}.
\]
Algorithm 7: INVERSEFREE2IPM - Inverse Free path following interior point method (IPM)

1 Input: Full column rank matrix \( A \in \mathbb{R}^{n \times d} \), initial feasible point \( x^{(0)} \), slack \( s^{(0)} \), centrality parameter \( t^{(0)} \), final centrality parameter \( t^{(end)} \) all with bit complexity \( \ell \).
2 Output: \( \hat{x} \in \mathbb{Q}^{n}_{\geq 0} \) such that \( A^\top \hat{x} = b \) and 
   \[ e^\top \hat{x} \leq \min_{x: A^\top x = b, x \geq 0} e^\top x + 1.1 \sqrt{n} \cdot t^{(end)} \].
3 Set \( \mu^{(0)} = X^{(0)} s^{(0)}, \beta = 10^4 \), and \( k = 1 \)
4 while \( t^{(k-1)} \geq t^{(end)} \) do
  5 Set \( \delta^{(k)} = -0.5 (\mu^{(k-1)} - 7^{(k-1)}) \)
  6 Set \( \overline{x}^{(k)} = \text{QRound}(x^{(k)}, 101, 0), \overline{s}^{(k)} = \text{QRound}(s^{(k)}, 101, 0), \delta^{(k)} = \text{QRound}(\delta^{(k)}, 1001, 0) \)
  7 Compute \( \delta^{(k)} = A (A^\top \overline{x}^{(k)} (\overline{s}^{(k)})^{-1} A)^{-1} A^\top (\overline{s}^{(k)})^{-1} \delta^{(k)} \) using Theorem 2.4.3.1
  8 Compute \( \overline{x}^{(k)} = (\overline{s}^{(k)})^{-1} \delta^{(k)} - \overline{x}^{(k)} (\overline{s}^{(k)})^{-1} \delta^{(k)} \)
  9 Update \( x^{(k)} = x^{(k-1)} + \text{Round}(\overline{s}^{(k)}, \exp(-4n)), \) \( s^{(k)} = s^{(k-1)} + \text{Round}(\overline{s}^{(k)}, \exp(-4n)) \), and \( t^{(k)} = \text{Round}((1 - \frac{1}{\beta \sqrt{n}}) t^{(k-1)}, \exp(-4n)) \)
10 For all \( i \in [n] \) such that \( x_i^{(k)} \leq \exp(-3n) \), set \( x_i^{(k)} = 0 \), remove its corresponding entry in \( c \) and corresponding row in \( A \)
11 Set \( k = k + 1 \)
12 return \((x^{(k-1)}, s^{(k-1)})\)

Now since \( A \delta_y + \delta_s = 0 \),
\[
-A^\top \overline{X} \overline{S}^{-1} A \delta_y = A^\top \overline{S}^{-1} \delta_y.
\]

Now multiplying \( A (A^\top \overline{X} \overline{S}^{-1} A)^{-1} \) from left, we have
\[
-A \delta_y = A (A^\top \overline{X} \overline{S}^{-1} A)^{-1} A^\top \overline{S}^{-1} \delta_y.
\]

Finally the result follows by noting that \( \delta_s = -A \delta_y \).

Remark 2.4.3.4. For \( i \in [n] \), we define \( \overline{x}_i \) and \( \overline{s}_i \) as the closest power of \( 10^1/100 \) to \( x_i \) and \( s_i \), respectively. Then \[ \left\| \overline{x} - x \right\|_\infty, \left\| \overline{s} - s \right\|_\infty \leq 0.01. \]

For the rest of the section, we prove the convergence of IPM when we use the approx-
imate values for taking a step. This is achieved by showing that: 1) \( x \) and \( s \) stay positive through the whole course of the algorithm (see Lemma 2.4.3.5); and 2) \( \|(x \odot s - \overline{7})/t\| \)
stays less than or equal to 0.01 (see Lemma 2.4.3.7). The latter implies that when \( t \) is small enough, we are close to the optimal objective value.

**Lemma 2.4.3.5.** Let \( t > 0, \mu = x \odot s, \delta_{\mu} = -h(\mu - t), \|\bar{x}-x\|_{\infty} \leq 0.01, \|\bar{s}-s\|_{\infty} \leq 0.01, \|
\frac{\delta_{\mu} - \delta_{\mu}}{\delta_{\mu}}\|_{\infty} \leq 0.01 \) and

\[
\bar{\delta}_{s} = A(A^{\top}\bar{X}S^{-1}A)^{-1}A^{\top}\bar{S}^{-1}\delta_{\mu} \quad \text{and} \quad \bar{\delta}_{x} = S^{-1}\delta_{\mu} - X\bar{S}^{-1}\delta_{s}.
\]

Moreover suppose \( \|\frac{\mu-t}{t}\|_{2} \leq 0.01. \) Then \( \|X^{-1}\bar{\delta}_{x}\|_{2}, \|S^{-1}\bar{\delta}_{s}\|_{2} \leq 0.15h. \)

**Proof.** First note that

\[
\|(x \odot s - \overline{7})/t\|_{\infty} \leq \|(x \odot s - \overline{7})/t\|_{2} \leq 0.01.
\]

Therefore \( 0.99 \cdot t \leq \|x \odot s\|_{\infty} \leq 1.01 \cdot t. \) Moreover since \( \|\bar{x}-x\|_{\infty}, \|\bar{s}-s\|_{\infty} \leq 0.01, \) for all \( i \in [n], \) \( 0.99x_{i} \leq \bar{x}_{i} \leq 1.01x_{i}, \) and \( 0.99s_{i} \leq \bar{s}_{i} \leq 1.01s_{i}. \) Thus

\[
\frac{99^{3}}{100^{3}}t \leq \|\overline{x} \odot \overline{s}\|_{\infty} \leq \frac{101^{3}}{100^{3}}t.
\]

Denoting \( \overline{P} := \overline{X}^{1/2}S^{-1/2}A(A^{\top}\overline{X}S^{-1}A)^{-1}A^{\top}\bar{S}^{-1/2}X^{1/2}, \) we have

\[
S^{-1}\bar{\delta}_{s} = \overline{X}X^{-1/2}S^{-1/2}P\bar{X}^{-1/2}S^{-1/2}\bar{\delta}_{\mu}.
\]

Now since \( \overline{P} \) is an orthogonal projection matrix, we have

\[
\|\bar{S}^{-1}\bar{\delta}_{s}\|_{2} = \|\overline{X}^{-1/2}S^{-1/2}P\bar{X}^{-1/2}S^{-1/2}\bar{\delta}_{\mu}\|_{2}
\leq \|\overline{X}^{1/2} \odot S^{-1/2}\|_{\infty} \|\bar{P}\|^{-1/2}\|S^{-1/2}\bar{\delta}_{\mu}\|_{2}
\leq \frac{100^{1.5}}{99^{1.5}} \cdot \frac{1}{\sqrt{t}} \|\bar{P}\|^{-1/2}\|S^{-1/2}\bar{\delta}_{\mu}\|_{2}
\]

85
Now we have
\[ \| S^{-1} \delta_s \|_2 \leq \frac{100^{1.5}}{99^{1.5}} \cdot \frac{1}{\sqrt{t}} \| X^{-1/2} S^{-1/2} \delta_{\mu} \|_2 \]
\[ \leq \frac{100^3}{99^3} \cdot \frac{1}{t} \| \delta_{\mu} \|_2 . \]

Now since \( \delta_{\mu} = -h(\mu - \tilde{\tau}) \) and \( \| (\mu - \tilde{\tau})/t \|_2 \leq 0.1 \), we have \( \| S^{-1} \delta_s \|_2 \leq 0.05h \). For \( X^{-1} \delta_s \), we similarly have
\[ \bar{X}^{-1} \delta_s = \bar{X}^{-1/2} S^{-1/2} (I - \bar{P}) \bar{X}^{-1/2} S^{-1/2} \delta_{\mu} . \]

Since \( (I - \bar{P}) \) is also a projection matrix, by a similar argument as above, we have
\[ \| X^{-1} \delta_s \|_2 \leq 0.05h. \]

\[ \square \]

**Remark 2.4.3.6.** Let \( h \leq 1 \). Then since \( \| X^{-1} \delta_s \|_\infty \leq \| X^{-1} \delta_x \|_2 \leq 0.15 \), we have for any entry \( i \in [n] \), \( \delta_{x}(i)/x(i) \leq 0.05 \). Therefore, \( x(i) + \delta_x(i) \geq 0.95x(i) > 0 \). A similar argument gives \( s(i) + \delta_s(i) > 0 \). Therefore the entries of \( x \) and \( s \) stay positive after an update.

**Lemma 2.4.3.7.** Let \( A \) be an integer matrix, \( \| \frac{S-x}{x} \|_\infty \), \( \| \frac{S-x}{x} \|_\infty \leq 0.01 \), \( \delta_{\mu} = -h(\mu_1 - \tilde{\tau}_1) \), and \( \mu_1 = x \odot s \), such that \( h = 0.1 \). Moreover let \( \delta_{\mu} \in \mathbb{R}^n \) such that and \( \| \delta_{\mu} \|_\infty \leq 0.001 \). Let
\[ \delta_s = A(A^\top S^{-1} A)^{-1} A^\top S^{-1} \delta_{\mu} , \] and \( \delta_x = S^{-1} \delta_{\mu} - S^{-1} S^{-1} \delta_{s} . \)

Moreover suppose \( \| \frac{\mu_1 - \tilde{\tau}_1}{t_1} \|_2 \leq 0.01 \). Let \( \mu_2 = (x + \delta_x) \odot (s + \delta_s) \) and \( t_2 = (1 - 1/(\beta \sqrt{n}))t_1 \), where \( \beta = 10^4 \). Then \( \| \frac{\mu_2 - \tilde{\tau}_2}{t_2} \|_2 \leq 0.01 \).
Proof. By triangle inequality,

\[
\frac{\mu_2 - \tilde{t}_2}{t_2} \leq \frac{(x + \delta_x + s + \delta_s - \tilde{t}_2)}{t_2} + \frac{(x - \tilde{x}) \circ \delta_s}{t_2} + \frac{(x - \tilde{s}) \circ \delta_x}{t_2} + \frac{\delta_x \circ \tilde{\delta}_s}{t_2} + \frac{\delta_s \circ \tilde{\delta}_x}{t_2}.
\]

Now by Lemma 2.4.4.1,

\[
\frac{\|\tilde{\delta}_x \circ \tilde{\delta}_s\|}{t_2} = \frac{\|\tilde{\delta}_x \circ \tilde{\delta}_s\|}{t_2} \leq \frac{\|x \circ s\|}{t_2} \|\tilde{\delta}_x\| \|S^{-1} \tilde{\delta}_s\| \\
\leq \frac{1.01}{1 - 1/(\beta \sqrt{n})} (0.05h)^2 \\
\leq 0.00003.
\]

Now denoting \(\tilde{P} := \tilde{X}^{-1/2} X^{-1/2} A(A^T X S^{-1} A)^{-1} A^T S^{-1/2} X^{-1/2}\), we have

\[
\tilde{x} \circ \delta_s + \tilde{s} \circ \delta_x = \tilde{X}^{-1/2} S^{-1/2} (I - \tilde{P}) S^{-1/2} \delta_s + \tilde{X}^{-1/2} S^{-1/2} \delta_s
\]

Therefore

\[
\frac{\|x \circ s - \tilde{t}_2 + \tilde{x} \circ \delta_s + \tilde{s} \circ \delta_x\|}{t_2} \leq \frac{1}{1 - 1/(\beta \sqrt{n})} \left( \frac{\|x \circ s - \tilde{t}_1 + \tilde{x} \circ \delta_s + \tilde{s} \circ \delta_x\|}{t_1} + \frac{1}{1 - 1/(\beta \sqrt{n})} \frac{1}{\beta \sqrt{n}} \right) \\
\leq \frac{1}{1 - 1/(\beta \sqrt{n})} \left( \frac{\|x \circ s - \tilde{t}_1 + \delta_s\|}{t_1} + \frac{1}{1 - 1/(\beta \sqrt{n})} \frac{\|\tilde{\delta}_s\|}{t_1} + 0.0002 \right)
\]

87
\[
\begin{align*}
&\leq \frac{1}{1 - 1/(\beta \sqrt{n})} \left\| \frac{x \odot s - \bar{r}_1 + \delta_{\mu}}{t_1} \right\|_2 + \frac{0.001}{1 - 1/(\beta \sqrt{n})} \left\| \delta_{\mu} \right\|_2 + 0.0002 \\
&\leq \frac{1}{1 - 1/(\beta \sqrt{n})} \cdot \left\| \frac{(1 - h)(\mu_1 - \bar{r}_1)}{\bar{r}_1} \right\|_2 + 0.0004 \\
&\leq 0.9001 \cdot \left\| \frac{\mu_1 - \bar{r}_1}{\bar{r}_1} \right\|_2 + 0.0004.
\end{align*}
\]

Moreover we have
\[
\left\| \frac{(x - \bar{x}) \odot \bar{\delta}_s}{t_2} \right\|_2 \leq \frac{0.01}{1 - 1/(\beta \sqrt{n})} \cdot \left\| \frac{x \odot \bar{\delta}_s}{t_1} \right\|_2
\]
\[
= \frac{0.01}{1 - 1/(\beta \sqrt{n})} \cdot \left\| \frac{XSS^{-1}\bar{\delta}_s}{t_1} \right\|_2 \leq \frac{0.01 \cdot 1.01}{1 - 1/(\beta \sqrt{n})} \cdot \left\| S^{-1}\bar{\delta}_s \right\|_2 \leq 0.0002,
\]
where the last inequality follows from Lemma 2.4.3.5. Similarly we can show \( \left\| \frac{(s - \bar{s}) \odot \bar{\delta}_x}{t_2} \right\|_2 \leq 0.0002 \). Combining all of the above and since \( \left\| \frac{\mu_1 - \bar{r}_1}{\bar{r}_1} \right\|_2 \leq 0.01 \), we have
\[
\left\| \frac{\mu_2 - \bar{r}_2}{t_2} \right\|_2 \leq 0.01.
\]

\[
\square
\]

We are now equipped to prove the main result of this section.

**Proof of Theorem 2.1.1.5.** For ease of notation, we drop the superscripts denoting iteration number for \( \bar{X}, \bar{S}, \bar{\delta}_\mu \) in this proof. First note that in each iteration of the IPM (Algorithm 7), we first compute
\[
\bar{\delta}_s = A(A^\top X S^{-1} A)^{-1} A^\top \bar{S}^{-1} \bar{\delta}_\mu,
\]
and then use this to compute \( \bar{\delta}_{s}^{(k)} = \bar{S}^{-1} \bar{\delta}_\mu - \bar{X}^{(k)} S^{-1} \bar{\delta}_s \). Note that by construction \( \bar{X}, \bar{S}, \bar{\delta}_\mu \) are rational matrices and vectors.

Now without loss of generality, we assume \( A \) is an integer matrix. Otherwise if the
bit complexity of $A$ is $\ell$, we multiply $A$ by $2^\ell$. This converts the matrix to an integer matrix. Note that this does not change the bit complexity of the matrix, and we later divide the solution of the linear system $(A^T\overline{X}S^{-1}A)z = A^T\overline{S}^{-1}\delta_\mu$ by $2^{2\ell}$. To find the value of $(A^T\overline{X}S^{-1}A)^{-1}A^T\overline{S}^{-1}\delta_\mu$, in each iteration, we find $z \in \mathbb{Q}^d$ such that $
abla_{101^s_1 \cdot 100^s_2} (A^T\overline{X}S^{-1}A)z = 101^s_1 \cdot 1000^s_3 A^T\overline{S}^{-1} \cdot \delta_\mu$, where $101^s_1$ is the denominator of the smallest entry of $1/\overline{s}$, $100^s_2$ is the denominator of the smallest entry of $x$, and $1000^s_3$ is the denominator of the smallest entry of $\overline{s}$. We then use this to find $\overline{\delta}_x, \overline{\delta}_s$, round these to a relative error of $\exp(-O(n))$ and add it to $x$ and $s$. This rounding introduces an error of size $\exp(-O(n))\kappa(A)$ in $A^T x = b$. Picking a large enough constant, we can make sure the total aggregate error over the whole course of the algorithm caused by this rounding is exponentially small.

Note that $s_1, s_2, s_3$ determine the bit complexity of the matrix and the vector of the corresponding integer linear systems we solve, and these numbers depend on the magnitude of smallest and largest entries of $x, s, \delta_\mu$.

First note that by definition $\delta_\mu = -0.5(\mu - \overline{\ell})$. Therefore since by Lemma 2.4.3.7, it is maintained that over the course of the algorithm $\| (\mu - \overline{\ell})/t \|_2 \leq 0.01$, for all $i \in [n]$,

$$\frac{0.99}{2} t \leq |(\delta_\mu)_i| \leq \frac{1.01}{2} t.$$  

Therefore over the whole course of the algorithm

$$\frac{0.99}{2} t^{(\text{end})} \leq |(\delta_\mu)_i| \leq \frac{1.01}{2} t^{(0)}.$$  

Now note that since for all $x$ such that $A^T x = b$ and $x \geq 0$, $\|x\|_2 \leq R$, for all $x$ such that $\|A^T x - b\| \leq \exp(-Cn)$ and $x \geq 0$, $\|x\|_2 \leq R + \exp(-(C - 2)n) \leq 2R$. Moreover since $\mu = x \odot s$ and $\| (\mu - \overline{\ell})/t \|_2 \leq 0.01$, $s_i \geq 0.99 t^{(\text{end})}/2R$. Now note that by construction $x_i$ is always either greater than or equal to $\exp(-3n)$ or is zero. Therefore $s_i \leq \exp(3n) \cdot 1.01 \cdot t^{(0)}$.

Now we need to bound these values for line 7 and line 9 of Algorithm 3. For line 7, we
have \( t^{(0)} = 2^{16} \varepsilon^{-3} n^2 \frac{R}{r} \cdot \|c\|_\infty R \) and \( t^{(\text{end})} = \|c\|_\infty R \). Therefore noting that \( \varepsilon = 1/(100 \sqrt{n}) \), the number of iterations is \( O(n^{0.5} \cdot \log(\frac{nR}{r})) \). Moreover the outer radius of the modified LP is less than \( 200(n+1) \sqrt{n} \cdot R \). Therefore by above arguments the bit complexity of right-hand side of linear systems we solve for line 7 of Algorithm 3 is bounded by

\[
O(n + \log(n \|c\|_\infty R/r)).
\]

Combining this with number of iterations and Theorem 2.4.3.1, bounds the running time of line 7 of Algorithm 3.

For line 9 of Algorithm 3, \( t^{(0)} \leq \frac{7}{6} \|c\|_\infty R \), and \( t^{(\text{end})} \) is \( \varepsilon / 2n \). Therefore the number of iterations is \( O(\sqrt{n} \log(n \|c\|_\infty R/\varepsilon)) \). This gives a bit complexity of \( O(n+\log(n \|c\|_\infty R/\varepsilon)) \) for the right-hand side of the linear systems solved for line 9 of Algorithm 3, and combining this with iteration number and Theorem 2.4.3.1 bounds the running time for this step. \( \square \)

2.4.4 Solving Sparse LP faster than Matrix Multiplication for \( \omega > 2.5 \)

The main result of this section is the following theorem achieved by Algorithm 8.

**Theorem 2.4.0.1.** \([\ell_2\text{-IPM for sparse matrices}]\) Let the matrix multiplication exponent \( \omega > 2.5 \). Given \( A \in \mathbb{R}^{n \times d} \) with full column-rank, \( b \in \mathbb{R}^d \), \( c \in \mathbb{R}^n \), \( 1 \leq m \leq n^{1/4} \) an error parameter \( 0 < \varepsilon < 1 \), and a linear program \( \min_{A^T x = b, x \geq 0} c^T x \) with inner radius \( r \) and outer radius \( R \), there exists an algorithm that finds \( \tilde{x} \in \mathbb{R}^n \) such that

\[
c^T \tilde{x} \leq \min_{A^T x = b, x \geq 0} c^T x + \varepsilon \cdot \|c\|_\infty R , \quad \text{and} \quad \|A^T \tilde{x} - b\|_2 \leq \varepsilon \cdot \|b\|_2 ,
\]

in time \( \tilde{O} \left( \left( \text{NNZ}(A) \cdot m^2 \cdot n + \frac{n^\omega}{m^{\omega - 3}} + n^{2.5} \right) \cdot \log^2 \left( \frac{\varepsilon \cdot \|b\|_2}{\varepsilon R} \right) \right) \) with high probability.

We define the vector \( \mu^{(k)} := x^{(k)} \odot s^{(k)} \). We want to maintain \( \|\mu^{(k)} - \tilde{\mu}^{(k)}\|_2 = O(t^{(k)}) \). The update is by taking the gradient of \( \|\mu^{(k)} - \tilde{\mu}^{(k)}\|_2 \), i.e., \( \mu^{(k+1)} = \mu^{(k)} + \delta^{(k)} \mu \) where
Lemma 2.4.4.1. Let
\[
\|X\|_F, \|X^{-1}\|_F, \|S\|_F, \|S^{-1}\|_F, \|A\|_F \leq \kappa.
\]  \tag{2.4.3}

Let
\[
\delta_x = \mathbf{X}^{rac{1}{2}} \mathbf{S}^{-rac{1}{2}} (\mathbf{I} - \mathbf{P}) \mathbf{X}^{-rac{1}{2}} \mathbf{S}^{-rac{1}{2}} \delta \mu, \quad \text{and} \quad \delta_s = \mathbf{X}^{-rac{1}{2}} \mathbf{S}^{rac{1}{2}} \mathbf{P} \mathbf{X}^{-rac{1}{2}} \mathbf{S}^{-rac{1}{2}} \delta \mu,
\]
where \( \mathbf{P} := \mathbf{X}^{rac{1}{2}} \mathbf{S}^{-rac{1}{2}} \mathbf{A} \mathbf{D}^{-1} \mathbf{S}^{-rac{1}{2}} \mathbf{X}^{rac{1}{2}} \), \( \delta \mu = -\mathbf{h} (\mu - \overline{\mu}) \), and \( \mu = \mathbf{x} \odot \mathbf{s} \) such that \( \left\| (\mathbf{A}^\top \mathbf{X} \mathbf{S}^{-1} \mathbf{A})^{-1} - \mathbf{D}^{-1} \right\|_F \leq \varepsilon \) such that \( \kappa^4 \varepsilon \leq 0.1 \). Moreover suppose \( \left\| \frac{\mu - \overline{\mu}}{\mathbf{t}} \right\|_2 \leq 0.1 \). Then
\[
\| \mathbf{X}^{-1} \delta_x \|_2, \| \mathbf{S}^{-1} \delta_s \|_2 \leq 0.15 h.
\]

**Proof.** First note that
\[
\left\| (\mathbf{x} \odot \mathbf{s} - \overline{\mathbf{t}}) / \mathbf{t} \right\|_\infty \leq \left\| (\mathbf{x} \odot \mathbf{s} - \overline{\mathbf{t}}) / \mathbf{t} \right\|_2 \leq 0.1.
\]

Therefore \( 0.9 \cdot \mathbf{t} \leq \| \mathbf{x} \odot \mathbf{s} \|_\infty \leq 1.1 \cdot \mathbf{t} \). We have
\[
\| \mathbf{S}^{-1} \delta_s \|_2 = \left\| \mathbf{X}^{-rac{1}{2}} \mathbf{S}^{-rac{1}{2}} \mathbf{P} \mathbf{X}^{-rac{1}{2}} \mathbf{S}^{-rac{1}{2}} \delta \mu \right\|_2 \leq \left\| \mathbf{X}^{-rac{1}{2}} \mathbf{S}^{-rac{1}{2}} \right\|_\infty \left\| \mathbf{P} \mathbf{X}^{-rac{1}{2}} \mathbf{S}^{-rac{1}{2}} \delta \mu \right\|_2 \leq \frac{10}{9} \cdot \frac{1}{\sqrt{\mathbf{t}}} \left\| \mathbf{P} \mathbf{X}^{-rac{1}{2}} \mathbf{S}^{-rac{1}{2}} \delta \mu \right\|_2.
\]

Now denoting \( \mathbf{P} := \mathbf{X}^{rac{1}{2}} \mathbf{S}^{-rac{1}{2}} \mathbf{A} (\mathbf{A}^\top \mathbf{X} \mathbf{S}^{-1} \mathbf{A})^{-1} \mathbf{A}^\top \mathbf{S}^{-rac{1}{2}} \mathbf{X}^{rac{1}{2}} \), since
\[
\left\| (\mathbf{A}^\top \mathbf{X} \mathbf{S}^{-1} \mathbf{A})^{-1} - \mathbf{D}^{-1} \right\|_F \leq \varepsilon,
\]
by (Equation 2.4.3), \( \| \mathbf{P} - \mathbf{P} \|_F \leq \kappa^4 \varepsilon \). Now by triangle inequality and since \( \mathbf{P} \) is an orthogonal projection matrix,
\[
\left\| \mathbf{P} \mathbf{X}^{-rac{1}{2}} \mathbf{S}^{-rac{1}{2}} \delta \mu \right\|_2 \leq \left\| \mathbf{P} \mathbf{X}^{-rac{1}{2}} \mathbf{S}^{-rac{1}{2}} \delta \mu \right\|_2 + \left\| (\mathbf{P} - \mathbf{P}) \mathbf{X}^{-rac{1}{2}} \mathbf{S}^{-rac{1}{2}} \delta \mu \right\|_2 \leq (1 + \kappa^4 \varepsilon) \left\| \mathbf{X}^{-rac{1}{2}} \mathbf{S}^{-rac{1}{2}} \delta \mu \right\|_2.
\]
Therefore

\[ \|S^{-1}\delta_s\|_2 \leq \frac{11}{9} \cdot \frac{1}{\sqrt{t}} \|X^{-1/2}S^{-1/2}\delta\|_2 \leq \frac{11}{9} \cdot \frac{1}{\sqrt{t}} \|X^{-1/2}S^{-1/2}\|_\infty \|\delta\|_2 \leq \frac{110}{81} \cdot \frac{1}{t} \|\delta\|_2 \]

Now since \( \delta = -h(\mu - \bar{\pi}) \) and \( \|(\mu - \bar{\pi})/t\|_2 \leq 0.1 \), we have \( \|S^{-1}\delta_s\|_2 \leq 0.15h \). For \( X^{-1}\delta_x \), we similarly have

\[ \|X^{-1}\delta_x\|_2 = \|X^{-1/2}S^{-1/2}(I - \bar{P})X^{-1/2}S^{-1/2}\delta\|_2 \leq \|X^{-1/2}S^{-1/2}\|_\infty \|(I - \bar{P})X^{-1/2}S^{-1/2}\delta\|_2 \leq \frac{10}{9} \cdot \frac{1}{\sqrt{t}} \|(I - \bar{P})X^{-1/2}S^{-1/2}\delta\|_2 . \]

Then a similar triangle inequality, and bound on \( \|(\mu - \bar{\pi})/t\|_2 \leq 0.1 \), gives

\[ \|X^{-1}\delta_x\|_2 \leq \frac{10}{9} \cdot \frac{1}{\sqrt{t}} (1 + \kappa^4 \epsilon) \|X^{-1/2}S^{-1/2}\|_\infty \|\delta\|_2 \leq 0.15h . \]

\[ \Box \]

**Note 2.4.4.2.** Let \( h \leq 1 \). Then since \( \|X^{-1}\delta_x\|_\infty \leq \|X^{-1}\delta_x\|_2 \leq 0.15 \), we have for any entry \( i \in [n] \), \( |\delta_x(i)/x(i)| \leq 0.15 \). Therefore, \( x(i) + \delta_x(i) \geq 0.85x(i) > 0 \). A similar argument gives \( s(i) + \delta_s(i) > 0 \). Therefore the entries of \( x \) and \( s \) stay nonnegative after an update.

**Lemma 2.4.4.3.** Let

\[ \|X\|_F, \|X^{-1}\|_F, \|S\|_F, \|S^{-1}\|_F, \|A\|_F \leq \kappa . \quad (2.4.4) \]
Let
\[
\delta_x = X^{1/2}S^{-1/2} (I - \overline{P})X^{-1/2}S^{-1/2} \delta_{\mu}, \text{ and } \delta_s = X^{-1/2}S^{1/2}P X^{-1/2}S^{-1/2} \delta_{\mu},
\]
where \(\overline{P} := X^{1/2}S^{-1/2} AD^{-1} A^\top S^{-1/2} X^{1/2},\) \(\delta_{\mu} = -h(\mu_1 - \overline{t}_1),\) and \(\mu_1 = x \odot s\) such that \(h \leq 0.5, \| (A^\top X S^{-1} A)^{-1} - D^{-1} \|_F \leq \varepsilon, \) and \(k^4 \varepsilon \leq 0.1.\) Moreover suppose \(\mu_2 = (x + \delta_x) \odot (s + \delta_s)\) and \(t_2 = (1 - 1/(1000 \sqrt{n})) t_1,\) and \(n \geq 10.\) Then \(\left\| \frac{\mu_2 - \overline{t}_2}{t_2} \right\|_2 \leq 0.1.\)

Proof. By triangle inequality,
\[
\left\| \frac{\mu_2 - \overline{t}_2}{t_2} \right\|_2 = \left\| \frac{(x + \delta_x) \odot (s + \delta_s) - \overline{t}_2}{t_2} \right\|_2 \leq \left\| \frac{x \odot s - \overline{t}_2}{t_2} \right\|_2 + \left\| \frac{x \odot \delta_s + s \odot \delta_x}{t_2} \right\|_2 + \left(\delta_x \odot \delta_s\right) \| t_2 \|_2.
\]

Now by Lemma 2.4.4.1,
\[
\left\| \frac{\delta_x \odot \delta_s}{t_2} \right\|_2 = \left\| \frac{XS X^{-1} \delta_x \odot S^{-1} \delta_s}{t_2} \right\|_2 \leq \left\| \frac{x \odot s}{t_2} \right\|_\infty \left\| X^{-1} \delta_x \right\|_\infty \left\| S^{-1} \delta_s \right\|_2 \leq \frac{1.1}{1 - 1/\sqrt{n}} (0.15h)^2 \leq 0.0125.
\]

Now denoting \(P := X^{1/2}S^{-1/2} A (A^\top X S^{-1} A)^{-1} A^\top S^{-1/2} X^{1/2},\)
\[
x \odot \delta_s + s \odot \delta_x = \delta_{\mu} + X^{1/2} S^{1/2} (P - \overline{P}) X^{-1/2} S^{-1/2} \delta_{\mu} + X^{1/2} S^{1/2} (\overline{P} - P) X^{-1/2} S^{-1/2} \delta_{\mu}
\]
\[
= \delta_{\mu} = -h(\mu_1 - \overline{t}_1).
\]

Therefore
\[
\left\| \frac{x \odot s - \overline{t}_2}{t_2} + \frac{x \odot \delta_s + s \odot \delta_x}{t_2} \right\|_2 \leq \frac{1}{1 - 1/\sqrt{n}} \left\| \frac{(1 - h)(\mu_1 - \overline{t}_1)}{\overline{t}_1} \right\|_2 + \frac{1}{1 - 1/\sqrt{n}} \left\| \frac{1}{1000 \sqrt{n}} \right\|_2
\]

94
Combining the above and using \( \| \frac{\mu_1 - \overline{\mu}}{\overline{\mu}} \| \) \( \leq 0.1 \), we have \( \| \frac{\mu_1 - \overline{\mu}}{\overline{\mu}} \| \leq 0.1 \).

**Lemma 2.4.4.4.** Let

\[
\|X\|_F, \|X^{-1}\|_F, \|S\|_F, \|S^{-1}\|_F, \|A\|_F \leq \kappa. \tag{2.4.5}
\]

In addition, let

\[
\delta_x = X^{1/2}S^{-1/2}(I - \overline{P})X^{1/2}S^{-1/2}\delta \mu, \quad \text{and} \quad \delta_s = X^{-1/2}S^{1/2}\overline{P}X^{1/2}S^{-1/2}\delta \mu,
\]

where \( \overline{P} := X^{1/2}S^{-1/2}AD^{-1}A^\top S^{-1/2}X^{1/2} \), \( \delta \mu = -h(\mu - \overline{\mu}) \), and \( \mu = x \otimes s \) such that \( \| (A^\top X S^{-1} A)^{-1} - D^{-1} \|_F \leq \varepsilon_1 \). Moreover suppose the initial feasible solution satisfies \( \| A^\top x^{(0)} - b \|_2 \leq \varepsilon_2 \). Then after \( k \) updates, we have \( A^\top (k) - b \|_2 \leq k k^2 \varepsilon_1 + \varepsilon_2 \).

**Proof.** By triangle inequality, we have

\[
\| A^\top (k) - b \|_2 \leq \| A^\top (0) - b \|_2 + \sum_{j=1}^{k} \| A^\top \delta x^{(k)} \|_2 \leq \varepsilon_2 + k k^2 \varepsilon_1.
\]

□

**Lemma 2.4.4.5.** Over the span of \( k \) iterations, Algorithm 8 makes at most \( O(k^2 \log n) \) changes to \( \overline{w} \).

**Proof.**

□

**Lemma 2.4.4.6.** Suppose an algorithm uses the sparse data structure of Theorem 2.3.2.2. Moreover assume that the algorithm runs for \( n^{1/2} \) iterations and after every \( k \) iterations, at most \( k^2 \) many entries are updated. Then for \( \omega > 2.5 \), the total running time of the
Proof. First note that the total contribution of terms of the form $\tilde{O}(\text{NNZ}(A) \cdot m^2 \cdot |S|)$ in the updates is $\text{NNZ}(A) \cdot m^2 \cdot n$ since there are at most $n$ updates over the course of the algorithm. So in the following, we omit the contribution of these terms. Moreover the total running time of query over the course of the algorithm is

$$
\tilde{O}(\text{NNZ}(A) \cdot m^2 \cdot n^0.5 + n^{2.5} \cdot \log^2 (\kappa / \varepsilon))
$$

Now note that the total cost of initialization and updates of rank more than $n/m$ is

$$
\tilde{O} \left( \left( d \cdot \text{NNZ}(A) \cdot m^{1.5} + \left( \frac{d}{m} \right)^{\omega} m^{2.5} \right) \log^2 (\kappa / \varepsilon) \right)
$$

because the number of such updates is at most $m^{0.5}$. Now note that the cost of an update of rank less than $n^\alpha$, is $O(n^2)$. Therefore the total cost of such updates over the course of the algorithm is $O(n^{2.5})$. For $k = 2^j$, the total cost of updates of rank at most $k^2$ (modulo $\log^2(\kappa / \varepsilon)$) is

$$
\left( n^2 \cdot (k^2)^{\omega-2} + \text{MM}(n, k^2, k^2) + k^{2\omega} \right) \cdot \frac{n^{0.5}}{k} \\
= n^{2.5} \cdot k^{2\omega-5} + \text{MM}(n, k^2, k^2) \cdot \frac{n^{0.5}}{k} + n^{0.5} \cdot k^{2\omega-1}
$$

Moreover since $\omega > 2.5$ and $k^2 \leq n/m$, we have

$$
n^{2.5} \cdot k^{2\omega-5} \leq n^{2.5} \left( \frac{n}{m} \right)^{(2\omega-5)/2} \leq \frac{n^{\omega}}{m^{\omega-2.5}}.
$$
Now since $3\omega - 1 > 0$, and $k^2 \leq n/m$,

$$n^{0.5} \cdot k^{2\omega-1} \leq n^{0.5} \cdot \left(\frac{n}{m}\right)^{(2\omega-1)/2} = \frac{n^{\omega}}{m^{\omega-0.5}}.$$  

We also have

$$\text{MM}(n, k^2, k^2) \cdot \frac{n^{0.5}}{k} \leq n^{\omega-2+0.5} \cdot k^3 \leq \frac{n^{\omega}}{m^{1.5}}.$$  

Moreover since $3 \geq \omega > 2.5$,

$$\frac{n^{\omega}}{m^{\omega-0.5}} \leq \frac{n^{\omega}}{m^{1.5}} \leq \frac{n^{\omega}}{m^{\omega-2.5}}.$$  

We can finally bound the cost of all updates by considering all powers of two between 1 and $(n/m)^{1/2}$ for $k$.  

\section*{2.5 $p$-Norm Regression}

In this section, we consider the problems of the following form.

$$\min_{x : Ax = b} \|x\|_p^p,$$  

where $A \in \mathbb{R}^{n \times d}$, $b \in \mathbb{R}^d$, and $p > 1$. We follow the approach of [8]. We first discuss a residual problem for (Equation 2.5.1) in Section 2.5.1. We show that (Equation 2.5.1) can be solved by solving $O_p(\alpha \log(n/e))$ instances of the residual problem to $\alpha$-approximation. We discuss how this residual problem can be turned into a mixed $(2, p)$-norm minimization problem with an extra linear constraint in Section 2.5.2. We can either directly optimize over this mixed $(2, p)$-norm problem (Section 2.5.5) or use a mixed $(2, \infty)$-norm problem as a proxy (Section 2.5.4). The latter approach introduces an extra factor of $n^{2/p}$ in the running time. However, since this approach is simpler, we first focus on this. Moreover, computing a constant factor approximation to mixed $(2, \infty)$-norm is of independent inter-
est. Both approaches require solutions to a series of weighted linear regression problems. Therefore before diving into either, we discuss the bit complexity of this weighted linear regression problem in Section 2.5.3. We extensively use the following inequality in this section.

**Fact 2.5.0.1 (Holder’s inequality).** Let \( \mathbf{x}, \mathbf{y} \in \mathbb{R}^n \) and \( p, q \in [1, \infty] \) such that \( \frac{1}{p} + \frac{1}{q} = 1 \). Then

\[
\| \mathbf{x} \odot \mathbf{y} \|_1 \leq \| \mathbf{x} \|_p \| \mathbf{y} \|_q .
\]

### 2.5.1 Residual Problem

We start this section by defining the smoothed \( p \)-norm function which was first introduced in [77] and has been used extensively in the \( p \)-norm minimization literature [8, 82, 83] since. We also refer to this function as a mixed \((2, p)\)-norm function because, under a certain threshold, it is a quadratic function and above the threshold, it is a power \( p \) function.

**Definition 2.5.1.1.** For \( p \geq 1 \) and a threshold \( t \in \mathbb{R}_{\geq 0} \), we define the (quadratically) smoothed \( p \)-norm function \( \gamma_p(t, \cdot) : \mathbb{R} \rightarrow \mathbb{R} \) as

\[
\gamma_p(t, x) := \begin{cases} 
\frac{p}{2} t^{p-2} x^2 & \text{if } |x| \leq t, \\
|x|^p + \left( \frac{p}{2} - 1 \right) t^p & \text{otherwise}. 
\end{cases} \tag{2.5.2}
\]

Overloading the notation, for a threshold vector \( t \in \mathbb{R}_{\geq 0}^n \), we define \( \gamma_p(t, \cdot) : \mathbb{R}^n \rightarrow \mathbb{R} \) as

\[
\gamma_p(t, \mathbf{x}) := \sum_{i=1}^n \gamma_p(t_i, x_i). \tag{2.5.3}
\]

The smoothed \( p \)-norm function gives a decent approximation for the Bregman divergence of the \( p \)-norm function. An important observation is that the smoothed \( p \)-norm function is symmetric (i.e., \( \gamma_p(t, x) = \gamma_p(t, -x) \)), while the Bregman divergence is not necessarily symmetric.

98
Lemma 2.5.1.2 ([8]). Let $p > 1$. Then for any $x, \Delta \in \mathbb{R}^n$,

$$
\|x\|_p^p - g^\top \Delta + \frac{p-1}{p \cdot 2^p} \gamma_p(|x|, \Delta) \leq \|x - \Delta\|_p^p \leq \|x\|_p^p - g^\top \Delta + 2^p \cdot \gamma_p(|x|, \Delta),
$$

where $g$ is the gradient of $p$-norm at $x$, i.e., $g = p \cdot |x|^{p-2} \odot x$.

Equipped with the above lemma, a natural approach is to take second-order Newton steps according to the smoothed $p$-norm function. In other words, we take steps according to the following residual problem.

Definition 2.5.1.3 (Residual problem). Given $x \in \mathbb{R}^n$ and $p > 1$, we define the mixed $(2, p)$-norm residual problem at $x$ as

$$
\arg \max_{\Delta^\top \Delta = 0} \ g^\top \Delta - \frac{p-1}{p \cdot 2^p} \cdot \gamma_p(|x|, \Delta),
$$

where $g = p \cdot |x|^{p-2} \odot x$ is the gradient of $\|x\|_p^p$.

To perform this second-order Newton approach, we require an initial point that is fairly close to the optimal. The next lemma states that the optimal solution to the quadratic problem is close to the optimal solution of the $p$-norm problem. This is similar to Lemma 4.8 of [8], but they only consider the case of $p \geq 2$ and the exact solution to the quadratic problem.

Lemma 2.5.1.4. Let $p > 1$, $\epsilon > 0$, $x^{(0)} \in \mathbb{R}^n$ such that $\|x^{(0)}\|_2 \leq (1 + \epsilon) \cdot \min_{x: A^\top x = b} \|x\|_2$, and $x^* = \arg \min_{x: A^\top x = b} \|x\|_p$. Then

$$
\|x^{(0)}\|_p^p \leq (1 + \epsilon)^p \cdot n^{p-2|p-2|/2} \|x^*\|_p^p.
$$

Proof. We have two cases. For $1 < p < 2$, we have $\frac{|p-2|}{2} = \frac{2-p}{2}$. Moreover $\|x\|_2 \leq \|x\|_p$. By taking $r = 2/p$ and $s = 2/(2 - p)$ for Holder’s inequality over vectors $v = [x_i^p]_i$, $u =$
\[ i \in R^n, \text{ respectively, we have } r, s \geq 1, \text{ and} \]

\[
\|x\|_p^p = \sum_{i=1}^{n} x_i^p \leq \|v\|_{2/p} \|u\|_{2/(2-p)} = \left(\sum_{i=1}^{n} x_i^2\right)^{p/2} n^{(2-p)/2} = n^{(2-p)/2} \|x\|_2^p.
\]

Therefore since by construction \(\|x(0)\|_2 \leq \|x^*\|_2\),

\[
\left\| x(0) \right\|_p^p \leq n^{(2-p)/2} \left\| x(0) \right\|_2^p \leq n^{(2-p)/2} \cdot (1 + \epsilon)^p \|x^*\|_2^p \leq n^{(2-p)/2} \cdot (1 + \epsilon)^p \|x^*\|_p^p.
\]

For \(p \geq 2\), \(\frac{p-2}{2} = \frac{p-2}{2}\), and \(\|x\|_p \leq \|x\|_2\). Taking \(r = p/2\), \(s = p/p - 2\) for Holder’s inequality over vectors \(v = [x_i^p]_i\), \(u = [1]_i \in R^n\), respectively, we have \(r, s \geq 1\), and

\[
\|x\|_2^2 = \sum_{i=1}^{n} x_i^2 \leq \|v\|_{p/2} \|u\|_{p/(p-2)} = \left(\sum_{i=1}^{n} x_i^p\right)^{2/p} n^{(p-2)/p} = n^{(p-2)/p} \|x\|_2^2.
\]

Thus,

\[
\left\| x(0) \right\|_p^p \leq \left\| x(0) \right\|_2^p \leq (1 + \epsilon)^p \|x^*\|_2^p \leq n^{(p-2)/2} \cdot (1 + \epsilon)^p \|x^*\|_p^p.
\]

\[ \square \]

We now show that by finding an approximate solution to the residual problem, we can move closer to the optimal. The following lemma, which is derived by Lemma 2.5.1.2 is useful for this purpose.

**Lemma 2.5.1.5** ([8]). Let \(p > 1\) and \(\lambda \leq \left(\frac{p-1}{p-4\epsilon}\right)^{1/\min\{1, p-1\}}\). Then for any \(x, \Delta \in R^n\),

\[
\|x\|_p^p - f(\lambda \Delta) \leq \|x - \lambda \Delta\|_p^p \leq \|x\|_p^p - \lambda f(\Delta),
\]

where \(f(\Delta) = g^\top \Delta - \frac{p-1}{p-2\epsilon} \cdot \gamma_p(|x|, \Delta).

Using this, we can prove the following lemma.

**Lemma 2.5.1.6** ([8]). Let \(\alpha, p > 1\), \(x, \hat{\Delta} \in R^n\) such that \(\hat{\Delta}\) is an \(\alpha\)-approximate solution to
the mixed $(2, p)$-norm residual problem at $\mathbf{x}$, i.e.,

$$
\mathbf{g}^\top \tilde{\mathbf{A}} - \frac{p - 1}{p \cdot 2^p} \cdot \gamma_p (|\mathbf{x}|, \tilde{\mathbf{A}}) \geq \frac{1}{\alpha} \cdot \max_{\mathbf{A}^\top \mathbf{A} = 0} \mathbf{g}^\top \mathbf{A} - \frac{p - 1}{p \cdot 2^p} \cdot \gamma_p (|\mathbf{x}|, \mathbf{A}).
$$

Then with $\lambda = \left(\frac{p - 1}{p \cdot 2^p}\right)^{1/\min(1, p - 1)}$ and $\text{opt} = \min_{\mathbf{x}: \mathbf{A}^\top \mathbf{x} = \mathbf{b}} \|\mathbf{x}\|_p^p$,

$$
\|\mathbf{x} - \lambda \tilde{\mathbf{A}}\|_p^p - \text{opt} \leq \left(1 - \frac{\lambda}{\alpha}\right) \cdot (\|\mathbf{x}\|_p^p - \text{opt}) .
$$

**Proof.** We define $f(\mathbf{A}) = \mathbf{g}^\top \mathbf{A} - \frac{p - 1}{p \cdot 2^p} \cdot \gamma_p (|\mathbf{x}|, \mathbf{A})$ and $\mathbf{A}^* = \arg \max_{\mathbf{A}^\top \mathbf{A} = 0} f(\mathbf{A})$. Then by Lemma 2.5.1.2

$$
f(\tilde{\mathbf{A}}) \geq \frac{1}{\alpha} f(\mathbf{A}^*) \geq \frac{1}{\alpha} f(\mathbf{x} - \mathbf{x}^*) \geq \frac{1}{\alpha} \left(\|\mathbf{x}\|^p_p - \|\mathbf{x}^*\|^p_p\right) = \frac{1}{\alpha} (\|\mathbf{x}\|^p_p - \text{opt}) .
$$

Moreover by Lemma 2.5.1.5,

$$
\|\mathbf{x} - \lambda \tilde{\mathbf{A}}\|_p^p - \text{opt} \leq \|\mathbf{x}\|^p_p - \lambda f(\tilde{\mathbf{A}}) - \text{opt} \leq - \frac{\lambda}{\alpha} (\|\mathbf{x}\|^p_p - \text{opt}) + \|\mathbf{x}\|^p_p - \text{opt}
\leq (1 - \frac{\lambda}{\alpha}) \cdot (\|\mathbf{x}\|^p_p - \text{opt}) .
$$

□

Then Lemmas 2.5.1.4 and 2.5.1.6 imply that Algorithm 9 finds an approximate solution.

Note that our algorithm considers the possible errors in solving the subproblems, e.g., the fact that the solution of $\mathbf{A}^\top \mathbf{x} = \mathbf{b}$ might not have a finite representation in fixed-point arithmetic and we have to have some error in our output.

**Theorem 2.5.1.7** (Iterative refinement for $p$-norm minimization). Algorithm 9 computes $\tilde{\mathbf{x}} \in \mathbb{R}^n$ such that

$$
\|\tilde{\mathbf{x}}\|^p_p \leq (1 + \epsilon) \|\mathbf{x}^*\|^p_p, \quad \text{and} \quad \|\pi_A \tilde{\mathbf{x}} - \mathbf{x}^*\|_2 \leq \epsilon \cdot \|\mathbf{x}^*\|_2 ,
$$
Algorithm 9: p-Norm Minimization by Approximately Solving a Series of Residual Problems

1. **Input:** Full column rank matrix $\mathbf{A} \in \mathbb{R}^{n \times d}$ ($n > d$), $\mathbf{b} \in \mathbb{R}^n$, $p > 1$, $\epsilon > 0$, $\alpha > 1$, where the residual problem can be solved to $\alpha$-approximation.
2. **Output:** $\mathbf{x} \in \mathbb{R}^n$ such that $\|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2 \leq \epsilon$ and $\|\mathbf{x}\|_p \leq (1 + \epsilon)\min_{x:A^\top x=b} \|x\|_p$.
3. Set $\lambda = \left(\frac{p-1}{p \cdot 4^p}\right)^{1/\min\{1,p-1\}}$ and $T = \left[\frac{\alpha \cdot \log(\frac{1+p \cdot n^{p-2}/2}{\epsilon})}{\lambda}\right]$.
4. Compute $\mathbf{x}^{(0)} \in \mathbb{R}^n$ such that $\|\mathbf{x}^{(0)}\|_2 \leq 1.1 \cdot \|\mathbf{x}^*\|_2$ and $\|\pi_{\mathbf{A}}\mathbf{x}^{(0)} - \mathbf{x}^*\|_2 \leq \frac{\epsilon}{2^T} \cdot \|\mathbf{x}^*\|_2$, where $\mathbf{x}^* = \arg\min_{x:A^\top x=b} \|x\|_2$.
5. for $t = 1, \ldots, T$ do
   6. Compute $\Lambda^{(t)} \in \mathbb{R}^n$ such that $\|\pi_{\mathbf{A}}\Lambda^{(t)}\|_2 \leq \frac{\epsilon}{2T \cdot \lambda} \cdot \|\mathbf{x}^*\|_2$ and $\mathbf{g}^\top \Lambda^{(t)} - \frac{p-1}{p \cdot 2^p} \gamma_p(\|\mathbf{x}^{(t-1)}\|_2, \Lambda^{(t)}) \leq \alpha \cdot \max_{\Lambda=0} \mathbf{g}^\top \Lambda - \frac{p-1}{p \cdot 2^p} \gamma_p(\|\mathbf{x}^{(t-1)}\|_2, \Lambda)$.
   7. Set $\mathbf{x}^{(t)} = \mathbf{x}^{(t-1)} - \lambda \Lambda^{(t)}$.
8. return $\mathbf{x} := \mathbf{x}^{(T)}$

in $O_p(\alpha \cdot \log(n/\epsilon))$ iterations, where $\mathbf{x}^* = \arg\min_{x:A^\top x=b} \|x\|_p$ and $\alpha$ is the approximation factor for solving the residual problem.

**Proof.** The number of iterations easily follows by line 3 of the algorithm and noting that $\lambda$ is a function of only $p$. Now since $\mathbf{x}^* = \mathbf{x}^{(0)} - \lambda \sum_{t=1}^{T} \Lambda^{(t)}$, by triangle inequality,

$$\|\pi_{\mathbf{A}}\mathbf{x}^* - \mathbf{x}^*\|_2 \leq \|\pi_{\mathbf{A}}\mathbf{x}^{(0)} - \mathbf{x}^*\|_2 + \lambda \sum_{t=1}^{T} \|\pi_{\mathbf{A}}\Lambda^{(t)}\|_2 \leq \frac{\epsilon}{2T} \cdot \|\mathbf{x}^*\|_2 + \lambda \cdot T \cdot \frac{\epsilon}{2T \cdot \lambda} \cdot \|\mathbf{x}^*\|_2 \leq \epsilon \cdot \|\mathbf{x}^*\|_2,$$

where the second inequality follows by construction of $\mathbf{x}^{(0)}$ and $\Lambda^{(t)}$ (see Algorithm 9).

Finally, since $1 - \frac{T}{\lambda} \leq \exp(-\frac{T}{\alpha})$, by Lemma 2.5.1.6,

$$\|\mathbf{x}^{(0)}\|_p - \|\mathbf{x}^*\|_p \leq \exp(-\frac{T}{\alpha}) \|\mathbf{x}^{(0)}\|_p.$$

Therefore since by Lemma 2.5.1.4, $\|\mathbf{x}^{(0)}\|_p \leq 1.1p \cdot n^{(p-2)/2} \|\mathbf{x}^*\|_p$, $\|\mathbf{x}\|_p - \|\mathbf{x}^*\|_p \leq \epsilon \cdot \|\mathbf{x}^*\|_p$, and the result follows. $\square$

The only remaining part of solving the $p$-norm minimization problem is to devise an
algorithm for solving the residual problem. We focus on this for the rest of the section.

2.5.2 Solving The Residual Problem

The objective of the residual problem (Definition 2.5.1.3) is a linear combination of a linear function and the smoothed $p$-norm function. We first discuss how the linear function can be removed from the objective and added as one of the constraints. This is essentially done by “guessing” the value of this linear term for the optimal solution.

Lemma 2.5.2.1 ([8]). Let $p > 1$, $x \in \mathbb{R}^n$, $b \in \mathbb{R}^d$, and $f : \mathbb{R}^n \to \mathbb{R}$ with $f(\Delta) = g^\top \Delta - \frac{p-1}{p} \gamma_p(|x|, \Delta)$, where $g = p |x|^{p-2} \odot x$ is the gradient of $||x||_p^p$. Moreover suppose $\Delta^* = \arg \max_{x : A^\top x = 0} f(\Delta)$ and $f(\Delta^*) \in [2^{j-1}, 2^j)$ for some $j \in \mathbb{Z}$. Let

$$\widehat{\Delta} = \arg \min_{\Delta} \gamma_p(|x|, \Delta)$$

$$\text{s. t. } g^\top \Delta = 2^{j-1},$$

$$A\Delta = 0.$$ (2.5.4)

For $\beta > 1$, let $\Delta^* \in \mathbb{R}^n$ such that $\gamma_p(|x|, \Delta) \leq \beta \cdot \frac{p}{p-1} \cdot 2^{j+p}$ and $g^\top \Delta^* \geq 2^{j-2}$. Then

1. $\gamma_p(|x|, \Delta) \leq \beta \cdot \frac{p}{p-1} \cdot 2^{j+p}$.

2. $f(\mu \Delta^*) \geq \frac{1}{8(4\beta p)^{1/(\min(p,2)-1)}} \cdot \frac{p-1}{p} \cdot f(\Delta^*)$ for $\mu = \left( \frac{1}{4\beta p} \right)^{1/(p-1)}$, if $1 < p \leq 2$, and

$$\mu = \frac{1}{8\beta},$$ otherwise.

The counterpart of Lemma 2.5.2.1 in [8] assumes we have $\Delta^*$ such that $g^\top \Delta^* = 2^{j-1}$ but since we cannot guarantee the existence of such a vector in fixed-point arithmetic, we replace this with the assumption that $g^\top \Delta^* \geq 2^{j-2}$. However, the proof is similar to that of [8] and only requires adjusting the constants. Lemma 2.5.2.1 implies that instead of approximately solving the residual problem, we can guess the interval $[2^{j-1}, 2^j)$ that contains the optimal objective value of the residual problem and approximately solve a problem of the form (Equation 2.5.4). Therefore to solve the residual problem, we need to
iterate over such intervals, compute an approximate solution of (Equation 2.5.4) for each, and take the one that achieves the maximum value for the function $f$. Now, the question is how many intervals we need to iterate over. The next lemma asserts that we only need to try a logarithmic number of intervals.

**Lemma 2.5.2.2** ([8]). Let $p > 1$ and $x^{(0)} = \arg \min_{A^\top x = b} \|x\|_2$. Moreover let $\bar{x} \in \mathbb{R}^n$ such that

$$\|\bar{x}\|_p^p > (1 + \epsilon) \min_{A^\top x = b} \|x\|_p^p.$$  

Let $f(\Delta) = g^\top \Delta - \frac{p-1}{p^2} \gamma_p(|\bar{x}|, \Delta)$ and $\lambda = \left(\frac{p-1}{p^2}\right)^{1/\min\{1,p-1\}}$. Then

$$\min_{A^\top \Delta = 0} f(\Delta) \in \left[\frac{\epsilon \|X^{(0)}\|_p^p}{n^{(|p-2|)/2}}, \frac{\|X^{(0)}\|_p^p}{\lambda}\right].$$

Lemma 2.5.2.2 asserts that if our current solution is not a $(1 + \epsilon)$-approximation, we only need to iterate over $\left[\log(\lambda n^{(p-2)/2}/\epsilon)\right] + 1$ intervals for (Equation 2.5.4) in order to approximately solve the residual problem. Moreover by substituting $\alpha = 16 \beta \cdot \frac{p}{p-1}$ from Lemma 2.5.2.1, in Algorithm 9, we have

$$T = \left\lfloor \frac{16 \beta \cdot p}{\lambda \cdot (p - 1)} \cdot \log\left(\frac{1.1^p \cdot n^{(p-2)/2}}{\epsilon}\right) \right\rfloor.$$ 

Then one can perform line 6 of Algorithm 9 by computing $\beta$-approximate solutions for $\left[\log(\lambda n^{(p-2)/2}/\epsilon)\right] + 1$ instances of problem (Equation 2.5.4) and taking the maximum. However, note that searching over such instances only improves the solution if the current solution is not a $(1 + \epsilon)$-approximation (see Lemma 2.5.2.2). Therefore in this approach, we need to add a conditional statement to the loop of Algorithm 9 to break and return $x^{(t-1)}$ if $\|x^{(t)}\|_p^p > \|x^{(t-1)}\|_p^p$.

Now, we need to approximately solve mixed $(2, p)$-norm minimization problems of the form (Equation 2.5.4). For the rest of the section, we focus on the case of $p \geq 2$. Our first approach is to solve such problems by approximately solving instances of a mixed
(2, ∞)-norm minimization problem.

The next lemma connects the smoothed $p$-norm function to a mixed $(2, \infty)$-norm function, which in turn allows us to approximately minimize the smoothed $p$-norm function, by approximately minimizing the mixed $(2, \infty)$-norm function.

**Lemma 2.5.2.3.** Let $\overline{\mathbf{A}} \in \mathbb{R}^{n \times d}$, $\overline{\mathbf{b}} \in \mathbb{R}^d$, $\mathbf{t} \in \mathbb{R}_{\geq 0}^n$, $p \geq 2$, $\overline{j} \in \mathbb{Z}$,

$$\widehat{\mathbf{\Delta}} = \arg \min_{\mathbf{\Delta} : \mathbf{A}^\top \mathbf{\Delta} = \overline{\mathbf{b}}} \gamma_p (\mathbf{t}, \mathbf{\Delta}),$$

and $\gamma_p (\mathbf{t}, \widehat{\mathbf{\Delta}}) \in [2^{\overline{j}-1}, 2^{\overline{j}})$. Let $\mathbf{r}, \mathbf{s} \in \mathbb{R}^n$, and $q \in \mathbb{Z}$ with $q \leq -2$, $\mathbf{r}_i = \frac{t_i^{p-2}}{2^{\max(\overline{j}, q+1)+2}}$, and $\mathbf{s}_i = \left(\frac{1}{2^{\max(\overline{j}, q+1)+2}}\right)^{1/p}$. Then

$$\min_{\mathbf{\Delta} : \mathbf{A}^\top \mathbf{\Delta} = \overline{\mathbf{b}}} \|\mathbf{\Delta}\|_2^2 + \|\mathbf{s} \odot \mathbf{\Delta}\|_\infty \leq 1,$$

and if $\overline{\mathbf{A}}$ such that $\|\overline{\mathbf{\Delta}}\|_2^2 + \|\mathbf{s} \odot \overline{\mathbf{\Delta}}\|_\infty \leq \theta$, then

$$\gamma_p (\mathbf{t}, \frac{1}{\theta} \cdot \overline{\mathbf{\Delta}}) \leq p \cdot \left(4\theta + 2^{p+1}\theta^p\right) \cdot \max\{\gamma_p (\mathbf{t}, \widehat{\mathbf{\Delta}}), 2^{\overline{j}}\}.$$

**Proof.** First, note that since $p \geq 2$, for $x, t \in \mathbb{R}$ and $t \geq 0$, $\gamma_p (t, x) \geq \max\{t^{p-2}x^2, |x|^p\}$. Therefore,

$$\sum_{i=1}^n t_i^{p-2} \overline{\mathbf{\Delta}}_i^2 + \left|\overline{\mathbf{\Delta}}_i\right|^p \leq 2\gamma_p (\mathbf{t}, \widehat{\mathbf{\Delta}}) < 2^{\overline{j}+1}.$$

Then by the construction of $\mathbf{r}, \mathbf{s}$ and since $t_i^{p-2} \overline{\mathbf{\Delta}}_i^2$ and $\left|\overline{\mathbf{\Delta}}_i\right|^p$ are nonnegative,

$$\|\overline{\mathbf{\Delta}}\|_2^2 \leq \frac{1}{2}, \text{ and } \|\mathbf{s} \odot \overline{\mathbf{\Delta}}\|_\infty \leq \|\mathbf{s} \odot \overline{\mathbf{\Delta}}\|_p \leq \frac{1}{2}.$$

Therefore

$$\min_{\mathbf{\Delta} : \mathbf{A}^\top \mathbf{\Delta} = \overline{\mathbf{b}}} \|\mathbf{\Delta}\|_2^2 + \|\mathbf{s} \odot \mathbf{\Delta}\|_\infty \leq \|\overline{\mathbf{\Delta}}\|_2^2 + \|\mathbf{s} \odot \overline{\mathbf{\Delta}}\|_\infty \leq 1.$$
Now since for \( x, t \in \mathbb{R} \) and \( |x| > t \geq 0 \), \( \frac{p}{2} |x|^p > |x|^p + (\frac{p}{2} - 1)t^p \), \( \gamma_p(t, x) \leq \frac{p}{2} (t^{p-2}x^2 + |x|^p) \).

Therefore,

\[
\gamma_p(t, n^{-1/p} \cdot \Delta) \leq \frac{p}{2} \sum_{i=1}^{n} t_i^{p-2}(n^{-1/p} \cdot \Delta_i)^2 + \frac{p}{2} \sum_{i=1}^{n} |n^{-1/p} \cdot \Delta_i|^p.
\]

We now bound the terms on the right-hand side. Since \( \|\Delta\|_r^2 + \|s \odot \Delta\|_\infty \leq \theta \),

\[
\sum_{i=1}^{n} \frac{t_i^{p-2}}{2^{\max\{j,q\}+2}} \cdot (n^{-1/p} \cdot \Delta_i)^2 \leq \sum_{i=1}^{n} \frac{t_i^{p-2}}{2^{\max\{j,q\}+2}} \cdot \Delta_i^2 = \|\Delta\|_r^2 \leq \theta.
\]

Moreover

\[
\left\| n^{-1/p} \Delta \right\|_p = 2^{(\max\{\tilde{j},q+1\}+p+1)/p} \left\| n^{-1/p} \cdot s \odot \Delta \right\|_p
\]

\[
\leq 2^{(\max\{\tilde{j},q+1\}+p+1)/p} \cdot n^{1/p} \left\| n^{-1/p} \cdot s \odot \Delta \right\|_\infty
\]

\[
\leq 2^{(\max\{\tilde{j},q+1\}+p+1)/p} \cdot \theta.
\]

Thus since \( \max\{\gamma_p(t, \tilde{\Delta}), 2^q\} \geq 2^{\max\{\tilde{j},q+1\} - 1} \),

\[
\gamma_p(t, n^{-1/p} \cdot \Delta) \leq \frac{p}{2} \left( 2^{\max\{\tilde{j},q\}+2} \theta + 2^{\max\{\tilde{j},q+1\}+p+1} \theta^p \right)
\]

\[
\leq p \cdot \left( 4 \theta + 2^{p+1} \theta^p \right) \cdot \max\{\gamma_p(t, \tilde{\Delta}), 2^q\}.
\]

\[\square\]

Lemma 2.5.2.3 bounds the values of entries of \( r \) and \( s \) if \( \tilde{j} \) is small. This combined with Lemma 2.5.2.1 implies that if \( \arg\max_{\Delta \succeq 0} g^\top \Delta - \frac{p-1}{p} \cdot \gamma_p(|x|, \Delta) < 2^j \), with \( \tilde{\Delta} = [A \mid g] \) and \( \tilde{\mathbf{b}} = \begin{bmatrix} \mathbf{0} \\ 2^{j-1} \end{bmatrix} \), we only need to try \( j + p - q + 1 + \log \left( \frac{p}{p-1} \right) \) values for \( r \) and \( s \) to find a vector \( \tilde{\Delta} \) with small \( \gamma_p \) value.

Although Lemma 2.5.2.3 implies that optimizing over the mixed \((2, \infty)\)-norm function
gives a vector $\overline{\Delta}$ with a small value for the $\gamma_p$ function, note that after multiplying $\overline{\Delta}$ by $n^{-1/p}$, the value of $g^\top \Delta$ decreases. Therefore we cannot use Lemma 2.5.2.1 to bound the value of the residual function for $n^{-1/p} \overline{\Delta}$. To obtain such a bound, we use the following lemma.

**Lemma 2.5.2.4 ([8]).** Let $p > 1$, $\lambda \geq 0$, and $t, \Delta \in \mathbb{R}^n$ with $t \geq 0$. Then

$$\min \{ \lambda^2, \lambda^p \} \gamma_p(t, \Delta) \leq \gamma_p(t, \lambda \Delta) \leq \max \{ \lambda^2, \lambda^p \} \gamma_p(t, \Delta).$$

Now by further scaling of $\overline{\Delta}$, we obtain a vector that gives a constant factor approximation for the residual problem. Note that by Lemma 2.5.2.1 and picking $q = \min\{j + p, -2\}$ in Lemma 2.5.2.3, we have $\gamma_p(t, n^{-1/p} \cdot \overline{\Delta}) \leq p \cdot (4\theta + 2p+1\theta^p) \cdot \frac{p}{p-1} \cdot 2^{j+p}$ assuming that the optimal value of the residual function is in $[2^{j-1}, 2^j)$. Moreover, the optimal value of the residual function is bounded by Lemma 2.5.2.2.

**Lemma 2.5.2.5.** Let $p \geq 2$, $x \in \mathbb{R}^n$, $b \in \mathbb{R}^d$, and $f : \mathbb{R}^n \to \mathbb{R}$ with $f(\Delta) = g^\top \Delta - \frac{p-1}{p^2} \gamma_p(|x|, \Delta)$, where $g = p |x|^{p-2} \odot x$ is the gradient of $||x||^p$. Moreover suppose $\Delta^* = \arg \max_{x: A^\top x = b} f(\Delta)$ and $f(\Delta^*) \in [2^{j-1}, 2^j)$ for some $j \in \mathbb{Z}$. Moreover let $\overline{\Delta} \in \mathbb{R}^n$ and $\theta \geq 1$ such that $g^\top \Delta \geq 2^{j-2}$ and $\gamma_p(t, n^{-1/p} \cdot \overline{\Delta}) \leq p \cdot (4\theta + 2p+1\theta^p) \cdot \frac{p}{p-1} \cdot 2^{j+p}$. Then for $\lambda = \frac{1}{8p \cdot (4\theta + 2p+1\theta^p)},$

$$f(n^{-2/p} \lambda \overline{\Delta}) \geq \frac{n^{-2/p}}{64p \cdot (4\theta + 2p+1\theta^p)} f(\Delta^*).$$

**Proof.** By Lemma 2.5.2.4 since $p \geq 2$ and $\lambda < 1,$

$$g^\top (n^{-2/p} \lambda \overline{\Delta}) - \frac{p-1}{p \cdot 2^p} \gamma_p(|x|, n^{-2/p} \lambda \overline{\Delta}) \geq n^{-2/p} \lambda \cdot 2^{j-2} - n^{-2/p} \lambda^2 \cdot \frac{p-1}{p \cdot 2^p} \cdot \gamma_p(t, n^{-1/p} \overline{\Delta})$$

$$\geq n^{-2/p} \lambda \cdot 2^{j-2} - n^{-2/p} \lambda^2 \cdot \frac{p-1}{p} \cdot 2^p \cdot (4\theta + 2p+1\theta^p) \cdot \frac{p}{p-1} \cdot 2^{j+p}$$

$$= n^{-2/p} \lambda \cdot 2^{j-2} - n^{-2/p} \lambda^2 \cdot p \cdot (4\theta + 2p+1\theta^p) 2^j$$

$$\geq \frac{n^{-2/p}}{8p \cdot (4\theta + 2p+1\theta^p)} (2^{j-2} - 2^{j-3})$$

107
Now that we established we can find a constant factor approximation for the residual problem by guessing the value of the linear term in the residual function and approximately solving the mixed \((2, p)\)-norm problem directly or by approximately solving the mixed \((2, \infty)\)-norm problem, we discuss how adding the linear constraint affects the condition number of our matrix. This is important since the bit complexity of inversion and inverse maintenance depends on the condition number of the matrix.

We show that the gradient term (arising from Taylor’s expansion of the \(p\)-norm — see Lemma 2.5.1.2) can be incorporated to the inverse because if the current solution is not close to the optimum of \(p\)-norm, the projection of the gradient vector into the kernel of matrix \(A\) is large, and therefore the matrix \(\overline{A} \coloneqq [A | g]\) does not have a large condition number. We first show that the projection is large.

\textbf{Lemma 2.5.2.6.} Let \(b \in \mathbb{R}^d\), and \(A \in \mathbb{R}^{n \times d}\) be a matrix with full column rank. Let \(p \geq 1\), \(g := p \cdot |\hat{x}|^{p-2} \odot \hat{x}\), be the gradient of \(\|x\|_p^p\) at \(\hat{x}\), and

\[
\n x^* = \arg \min_{x : A^\top x = b} \|x\|_p^p.
\]

Let \(0 < \varepsilon < 1\), and \(\hat{x} \in \mathbb{R}^d\) such that \(\|A^\top \hat{x} - b\|_2 \leq \frac{\varepsilon}{2\kappa^3}\varepsilon\) and \(\|\hat{x}\|_p^p > (1 + \varepsilon)\|x^*\|_p^p\). Let \(\kappa > 1\), \(\|x^*\|_p^p \geq 1/\kappa\), and

\[
\|A^\top A\|_F \cdot \|(A^\top A)^{-1}\|_F \cdot \|g\|_2 \cdot \|x^* - \hat{x}\|_2 \leq \kappa.
\]

Then \(\|(I - A(A^\top A)^{-1}A^\top)g\|_2 \geq \frac{\varepsilon}{2\kappa^2}\).
Proof. Since \( \|x\|_p^p \) is a convex function, we have

\[
(1 + \varepsilon) \|x^*\|_p^p + g^T (x^* - \tilde{x}) < \|\tilde{x}\|_p^p + g^T (x^* - \tilde{x}) \leq \|x^*\|_p^p.
\]

Therefore

\[
\varepsilon \|x^*\|_p^p \leq g^T (\tilde{x} - x^*).
\]

We have

\[
(\tilde{x} - x^*) = (I - A(A^T A)^{-1} A^T)(\tilde{x} - x^*) + A(A^T A)^{-1} A^T (\tilde{x} - x^*)
\]

Since \( A^T x^* = b \), \( \|A^T (\tilde{x} - x^*)\|_2 \leq \frac{\varepsilon}{2\kappa^{1.5}}. \) Therefore by Cauchy-Schwarz and triangle inequalities,

\[
\varepsilon \frac{1}{\kappa} \leq \|g^T (I - A(A^T A)^{-1} A^T)\|_2 \|\tilde{x} - x^*\|_2 + \|g^T\|_2 \|A(A^T A)^{-1}\|_2 \|A^T (\tilde{x} - x^*)\|_2 \leq \|g^T (I - A(A^T A)^{-1} A^T)\|_2 \kappa + \kappa^{2.5} \cdot \frac{\varepsilon}{2\kappa^{3.5}}
\]

Therefore

\[
\|g^T (I - A(A^T A)^{-1} A^T)\|_2 \geq \frac{\varepsilon}{2\kappa^2}.
\]

The next lemma states that if we add a new column \( g \) to the matrix \( A \) forming the matrix \( \overline{A} = [A|g] \), given that the projection of \( g \) into the kernel of \( A \) is not small, the condition number of \( \overline{A} \) is small.

**Lemma 2.5.2.7.** Let \( A \in \mathbb{R}^{n \times d} \), \( n > d \), be a matrix with full column rank. Moreover let \( g \in \mathbb{R}^n \). Suppose \( \kappa > 1 \), and

\[
\|A^T A\|_F, \|A^T A\|^{-1}_F, \|g\|_2, 1/\|(I - A(A^T A)^{-1} A^T)g\|_2 \leq \kappa.
\]
Then \( \| \overline{A}^T \overline{A} \|_F, \| (\overline{A}^T \overline{A})^{-1} \|_F \leq 8\kappa^7 \), where \( \overline{A} = \begin{bmatrix} A | g \end{bmatrix} \).

**Proof.** First note that \( g \) is not in the range of \( A \), since if \( g = Ay \), then

\[
\|(I - A(A^TA)^{-1}A^T)g\|_2 = \|Ay - A(A^TA)^{-1}A^T Ay\|_2 = 0,
\]

which is in contrast with the assumption. Therefore \( \overline{A} \) has full column rank and \( \overline{A}^T \overline{A} \) is invertible. Now note that

\[
\overline{A}^T \overline{A} = \begin{bmatrix}
A^T A & A^T g \\
g^T A & g^T g
\end{bmatrix}
\]

Therefore by triangle inequality and consistency of the Frobenius norm.

\[
\| \overline{A}^T \overline{A} \|_F \leq \| A^T A \|_F + 2 \| A^T g \| + \| g \|_2^2 \leq 4\kappa^2 \leq 8\kappa^7.
\]

Let \( s := \| g \|_2^2 - g^T A(A^T A)^{-1}A^T g \) be the Schur complement of \( \overline{A}^T \overline{A} \). By matrix inversion lemma, since \( A^T A \) and \( \overline{A}^T \overline{A} \) are invertible, \( s \) is also invertible and

\[
(\overline{A}^T \overline{A})^{-1} = \begin{bmatrix}
(A^T A)^{-1} + \frac{(A^T A)^{-1}A^T g g^T (A^T A)^{-1}}{s} & \frac{(A^T A)^{-1}A^T g}{s} \\
-\frac{g^T (A^T A)^{-1}}{s} & \frac{1}{s}
\end{bmatrix}.
\]

Now note that

\[
\|(I - A(A^T A)^{-1}A^T)g\|_2^2 = g^T (I - A(A^T A)^{-1}A^T) (I - A(A^T A)^{-1}A^T)g
\]

\[
= \| g \|_2^2 - 2 \cdot g^T A(A^T A)^{-1}A^T g
\]

\[
+ g^T A(A^T A)^{-1}A^T A(A^T A)^{-1}A^T g
\]

\[
= \| g \|_2^2 - g^T A(A^T A)^{-1}A^T g
\]

\[
= s.
\]

Therefore by assumption \( 1/s \leq \kappa^2 \). Now by the triangle inequality and the consistency of
the Frobenius norm, we have

$$\left\| (\mathbf{A}^\top \mathbf{A})^{-1} \right\|_F \leq \left\| (\mathbf{A}^\top \mathbf{A})^{-1} + \frac{(\mathbf{A}^\top \mathbf{A})^{-1} \mathbf{A} \mathbf{g} \mathbf{g}^\top \mathbf{A} (\mathbf{A}^\top \mathbf{A})^{-1}}{s} \right\|_F + 2 \cdot \left\| \frac{(\mathbf{A}^\top \mathbf{A})^{-1} \mathbf{A} \mathbf{g}}{s} \right\|_F + \frac{1}{s}$$

$$\leq \left\| (\mathbf{A}^\top \mathbf{A})^{-1} \right\|_F + \kappa^2 \left( \left\| (\mathbf{A}^\top \mathbf{A})^{-1} \mathbf{A} \mathbf{g} \right\|_F^2 + 2 \cdot \left\| (\mathbf{A}^\top \mathbf{A})^{-1} \mathbf{A} \mathbf{g} \right\|_F + 1 \right)$$

$$\leq \kappa + \kappa^2 \left( \left\| (\mathbf{A}^\top \mathbf{A})^{-1} \mathbf{A} \mathbf{g} \right\|_F + 1 \right)^2 \leq \kappa + \kappa^2 (\kappa^{2.5} + 1)^2 \leq 8 \kappa^7.$$

\[ \square \]

2.5.3 Weighted Linear Regression with Equality Constraints

In this section, we examine computing a high-accuracy solution to a weighted constrained linear regression problem using an erroneous inverse of a preconditioner. The inverse has error because we are working under the fixed-point arithmetic. To approximately solve the mixed \((2, p)-\)norm minimization problem, or the mixed \((2, \infty)-\)norm minimization problem, we need to solve \(\hat{O}_p(n^{1/3})\) such weighted constrained linear regression problem. We later discuss that for these problems, using inverse maintenance techniques, we can maintain an erroneous constant-factor spectral approximation of the inverse as the preconditioner.

Note that when solving the problem \(\arg\min_{\mathbf{x}: \mathbf{A}^\top \mathbf{x} = \mathbf{b}} \frac{1}{2} \left\| \mathbf{x} \right\|_W^2\), we require the error of the solution to be small in two different norms: the norms defined on matrices \(\mathbf{W}\) and \(\pi_\mathbf{A}\). Interestingly, as we see in the next lemma, \(\left\| \mathbf{x}^* \right\|_2\) is within a factor \(R\) of \(\left\| \pi_\mathbf{A} \mathbf{x}^* \right\|_2\). Therefore, we do not need \(\log(\kappa(\mathbf{A}))\) iterations of Richardson to achieve this.

**Lemma 2.5.3.1.** Let \(\mathbf{A} \in \mathbb{R}^{n \times d}\) with full column rank, \(\mathbf{b} \in \mathbb{R}^d\), \(\mathbf{W} \in \mathbb{R}^{n \times n}\) be a diagonal matrix with \(\mathbf{R} \mathbf{I} \succeq \mathbf{W} \succeq \mathbf{I}\), and

\[ \mathbf{x}^* := \arg\min_{\mathbf{x}: \mathbf{A}^\top \mathbf{x} = \mathbf{b}} \frac{1}{2} \left\| \mathbf{x} \right\|_W^2. \]
Then

\[ \|x^*\|_2 \leq R \cdot \|\pi Ax^*\|_2. \]

Proof. By Lemma 2.1.2.3, \( x^* = W^{-1} A (A^T W^{-1} A)^{-1} b \). Therefore since \( W^{-1} \preceq I \),

\[ \|x^*\|_2 \leq b^T (A^T W^{-1} A)^{-1} A^T W^{-2} A (A^T W^{-1} A)^{-1} b \]
\[ \leq b^T (A^T W^{-1} A)^{-1} A^T W^{-1} A (A^T W^{-1} A)^{-1} b \]
\[ = b^T (A^T W^{-1} A)^{-1} b. \]

Moreover

\[ \|\pi Ax^*\|_2 = x^*^T A (A^T A)^{-1} A^T x^* \]
\[ = b^T (A^T W^{-1} A)^{-1} A^T W^{-1} A (A^T A)^{-1} A^T W^{-1} A (A^T W^{-1} A)^{-1} b \]
\[ = b^T (A^T A)^{-1} b. \]

Now note that since \( \frac{1}{R} I \preceq W^{-1} \), we have \( \frac{1}{R} A^T A \preceq A^T W^{-1} A \). Therefore \( R (A^T A)^{-1} \succeq (A^T W^{-1} A)^{-1} \). Thus

\[ R \cdot \|\pi Ax^*\|_2 = R \cdot b^T (A^T A)^{-1} b \geq b^T (A^T W^{-1} A)^{-1} b \geq \|x^*\|_2. \]

\[ \square \]

We are now equipped to prove the main result of this subsection, which is the main subprocedure for both mixed \((2, \infty)\)-norm minimization and mixed \((2, p)\)-norm minimization.

**Lemma 2.1.2.3** (High-accuracy solutions for constrained weighted linear regression). Let \( A \in \mathbb{R}^{n \times d} \) have full column rank, \( b \in \mathbb{R}^d \), and \( W \in \mathbb{R}^{n \times n} \) be a diagonal matrix with \( RI \succeq W \preceq I \). Moreover let \( x^* = \arg \min_{x: A^T x = b} \frac{1}{2} \|x\|^2_W \). Then

\[ x^* = W^{-1} A (A^T W^{-1} A)^{-1} b. \]
Moreover given a matrix $\tilde{M}^{-1}$ such that there exists matrix $M$ with $\|\tilde{M}^{-1} - M^{-1}\|_F \leq \frac{\epsilon}{d.A.\|A^T W^{-1} A\|_2}$ and $A^T W^{-1} A \preceq M \preceq \lambda A^T W^{-1} A$ with a constant $\lambda \geq 1$, there is an algorithm that finds $\hat{x}$ such that

$$\|\hat{x} - x^*\|_2 \leq \epsilon \cdot \|x^*\|_2, \quad \|\hat{x}\|_W \leq (1 + \epsilon) \|x^*\|_W,$$

and

$$\|\pi_A(\hat{x} - x^*)\|_2 \leq \epsilon \|\pi_A x^*\|_2,$$

where $\pi_A$ is the projection matrix of matrix $A$, in $O((d^2 + \text{NNZ}(A)) \cdot \log(\kappa(A) \cdot R) \cdot \log^2(\frac{\epsilon}{\delta}))$ time.

**Proof.** Note that the gradient of $\frac{1}{2} \|x\|_W^2$ is $Wx$ and for any $x$ in the kernel of $A^T$, $A^T(x^* + x) = b$. Therefore $Wx^*$ should be orthogonal to the kernel of $A^T$. Therefore there exists $y$ such that $Ay = Wx^*$. Therefore $x^* = W^{-1} Ay$ and $b = A^T x^* = A^T W^{-1} Ay$. Solving for $y$, we have $y = (A^T W^{-1} A)^{-1} b$. Thus by $Ay = Wx^*$, we have $x^* = W^{-1} A(A^T W^{-1} A)^{-1} b$.

Since $A^T W^{-1} A$ is full-rank, $(A^T W^{-1} A)^{-1} b$ corresponds to a linear system of the form $(A^T W^{-1} A) z = b$. Therefore by using Richardson’s iteration (Lemma 2.2.0.2) as $z^{(k+1)} = z^{(k)} - \tilde{M}^{-1}(A^T W^{-1} A z^{(k)} - b)$ with $z^{(0)}$ and $\|\tilde{M}^{-1} - M^{-1}\|_F \leq \frac{\epsilon}{d.A.\|A^T W^{-1} A\|_2}$, we can guarantee that

$$\|z^{(k)} - z^*\|_M \leq (1 - \lambda^{-1} + \epsilon)^k \|z^*\|_M,$$

where $z^* = (A^T W^{-1} A)^{-1} b$. Since $x^* = W^{-1} A z^*$, we have

$$\left\|\begin{array}{c} z^{(k)} \\ z^* \end{array} \right\|_{A^T W^{-1} A} \leq \lambda \cdot (1 - \lambda^{-1} + \epsilon)^k \left\|\begin{array}{c} z^* \\ z^* \end{array} \right\|_{A^T W^{-1} A}.$$

Setting $x^{(k)} = W^{-1} A z^{(k)}$, we have

$$\left\|\begin{array}{c} z^{(k)} \\ z^* \end{array} \right\|_{A^T W^{-1} A}^2 = (z^{(k)} - z^*)^T A^T W^{-1} A (z^{(k)} - z^*)$$

$$= (z^{(k)} - z^*)^T A^T W^{-1} W W^{-1} A (z^{(k)} - z^*)$$

$$= \left\|x^{(k)} - x^*\right\|_W^2.$$
Similarly,
\[
\|z^*\|_{A^TW^{-1}A}^2 = (z^*)^T A^TW^{-1}Az^* = (z^*)^T A^TW^{-1}WW^{-1}Az^* = \|x^*\|_W^2.
\]

Thus
\[
\|x^{(k)} - x^*\|_W \leq \lambda \cdot (1 - \lambda^{-1} + \epsilon)^k \|x^*\|_W.
\]

(2.5.5)

Therefore, taking \( k > \frac{1}{\lambda - 1 - \epsilon} \log(R^2 \cdot \lambda/\epsilon) \), since \( RI \succeq W \succeq I \), we have
\[
\|x^{(k)} - x^*\|_2 \leq \frac{\epsilon}{R^2} \cdot \|x^*\|_W \leq \frac{\epsilon}{R} \cdot \|x^*\|_2.
\]

Thus by triangle inequality,
\[
\|x^{(k)}\|_W \leq (1 + \epsilon) \|x^*\|_W.
\]

Moreover, since projection only decreases the length of a vector and by Lemma 2.5.3.1, we have
\[
\|\pi_A(x^{(k)} - x^*)\|_2 \leq \|x^{(k)} - x^*\|_2 \leq \frac{\epsilon}{R} \cdot \|x^*\|_2 \leq \epsilon \cdot \|\pi_Ax^*\|_2.
\]

\( \square \)

The multiplicative weights update algorithms that we employ in the next two sections are susceptible to error (as opposed to interior point methods). More specifically, they require high-accuracy solutions to the weighted linear regression problems in the sense that if we output \( \hat{x} \in \mathbb{R}^n \) for the problem \( x^* = \arg \min_{x:A^Tx = b} \frac{1}{2} \|x\|_W^2 \), we need \( \|\hat{x} - x^*\|_2 \leq \frac{1}{\text{poly}(n)} \). This is required to guarantee certain potential functions are increasing rapidly and is implied by our iterative method with preconditioning for solving the weighted linear regression problems by taking an appropriate error parameter (Richardson’s iteration of Lemma 2.1.2.3).
2.5.4 Mixed $(2, \infty)$-Norm Minimization

In this section, we discuss a multiplicative weights update approach to find a constant-factor approximation to the weighted mixed $(2, \infty)$-norm problem. This problem is of independent interest, but as shown in Section 2.5.2, it can also be used to find a high-accuracy solution to the $p$-norm minimization problem.

**Theorem 2.1.1.9** (continuing from p. 32). Let $A \in \mathbb{R}^{n \times d}$ and $b \in \mathbb{R}^d$, $r, s \in \mathbb{R}^n_{\geq 0}$, $n \geq d$, such that the condition number of $A$ is less than $\kappa$ and the bit complexity of $r, s$, and $b$ are bounded by $\log(\kappa)$. For $0 < \epsilon < 1$, and $m \leq n^{(\omega-2)/(\omega-1)}$, there is an algorithm that outputs $\widehat{x}$ such that $\|\pi_A(\widehat{x} - x^*)\|_2 \leq \epsilon \|\pi_Ax^*\|_2$ and

$$
\|\widehat{x}\|_r^2 + \|s \odot \widehat{x}\|_\infty = O(1) \cdot (\|x^*\|_r^2 + \|s \odot x^*\|_\infty),
$$

where $x^* = \arg \min_{x: A^T x = b} \|x\|_r^2 + \|s \odot x\|_\infty$, in time

$$
\tilde{O}_p((n^\omega + n^{7/3} \cdot \log^2(1/\epsilon)) \log(\alpha_2 \kappa / \epsilon) \log(\alpha_1 \kappa) \log(\kappa / \epsilon)),
$$

where $\alpha_1 = 1/(\min_{i \in [n]} r_i + s_i^2)$ and $\alpha_2 = (\max_{i \in [n]} r_i + \max_{i \in [n]} s_i) / \min_{i \in [n]} r_i$. Moreover, for sparse matrices, there is an algorithm that returns an output with the same guarantees with probability at least $1 - n^{-10} \cdot \log(\alpha_2)$ in time

$$
\tilde{O}
\left(
n^\omega m^{7/3-\omega} + \text{NNZ}(A) \cdot m^{4/3} \cdot n \cdot \log^2(1/\epsilon) + n^{7/3} \cdot \log^2(1/\epsilon)
\right)
\cdot \log(\alpha_2 \kappa / \epsilon) \log(\alpha_1 \kappa) \log^2(\kappa / \epsilon)
\right).
$$

In this section we present our multiplicative weight update algorithm to solve a mixed $(2, \infty)$-norm problem — see Algorithms 10 and 11.

We first show that a certain weighted linear regression problem is related to the mixed $(2, \infty)$-norm minimization, and bounds on the optimum value of the mixed norm give sev-
eral bounds on the optimum solution of the weighted linear regression problem. Such a weighted linear regression problem is solved in every iteration of our multiplicative weights update algorithm (Algorithms 10 and 11).

**Lemma 2.5.4.1.** Let \( \mathbf{r}, \mathbf{s} \in \mathbb{R}^{n}_{\geq 0} \), \( \mathbf{A} \in \mathbb{R}^{n \times d} \), and \( \mathbf{b} \in \mathbb{R}^{d} \). Let \( \mathbf{w} \in \mathbb{R}^{n}_{\geq 0} \), and for all \( i \in [n] \), define \( \tilde{w}_i = \| \mathbf{w} \|_1 \cdot \mathbf{r}_i + \left( \mathbf{w}_i + \frac{1}{n} \| \mathbf{w} \|_1 \right) \cdot s_i^2 \) (similar to line 2 of Algorithm 11). Let

\[
\Lambda = \arg \min_{\Lambda: \mathbf{A}^\top \Lambda = \mathbf{b}} \| \Lambda \|_F^2 + \| \mathbf{s} \odot \Lambda \|_\infty, \quad \text{and} \quad \Lambda = \arg \min_{\Lambda: \mathbf{A}^\top \Lambda = \mathbf{b}} \| \Lambda \|_\infty^2.
\]

and suppose

\[
\| \Lambda \|_F^2 + \| \mathbf{s} \odot \Lambda \|_\infty \leq 1. \tag{2.5.7}
\]

Then

1. \( \| \Lambda \|_w^2 \leq 6 \cdot \| \mathbf{w} \|_1 \).
2. \( \| \Lambda \|_r^2 \leq 6 \).
3. \( \sum_{i=1}^{n} \mathbf{w}_i \left| s_i \tilde{\Lambda}_i \right| \leq \sqrt{6} \| \mathbf{w} \|_1 \).
4. For all \( i \in [n] \), \( s_i \tilde{\Lambda}_i \leq \sqrt{6n} \).

**Proof.** By definition of \( \tilde{\mathbf{w}} \), we have

\[
\| \tilde{\Lambda} \|_w^2 = \min_{\Lambda: \mathbf{A}^\top \Lambda = \mathbf{b}} \sum_{i=1}^{n} \left( \| \mathbf{w} \|_1 \cdot \mathbf{r}_i + \left( \mathbf{w}_i + \frac{1}{n} \| \mathbf{w} \|_1 \right) s_i^2 \right) \tilde{\Lambda}_i^2 \tag{2.5.8}
\]

\[
\leq \sum_{i=1}^{n} \left( \| \mathbf{w} \|_1 \cdot \mathbf{r}_i + \left( \mathbf{w}_i + \frac{1}{n} \| \mathbf{w} \|_1 \right) s_i^2 \right) \tilde{\Lambda}_i^2 \tag{2.5.9}
\]

By assumption (Equation 2.5.7) and definition of \( \tilde{\Lambda} \),

\[
\sum_{i=1}^{n} \| \mathbf{w} \|_1 \cdot \mathbf{r}_i \tilde{\Lambda}_i^2 \leq \| \mathbf{w} \|_1. \tag{2.5.10}
\]
Moreover
\[ \sum_{i=1}^{n} (w_i + \frac{1}{n} \|w\|_1) s_i^2 \cdot \Delta_i^2 = \sum_{i=1}^{n} w_i s_i^2 \cdot \Delta_i^2 + \frac{1}{n} \|w\|_1 \sum_{i=1}^{n} s_i^2 \cdot \Delta_i^2 \]

Since \( \|s \odot \Delta\|_{\infty} \leq 1 \), each \( s_i^2 \cdot \Delta_i^2 \leq 1 \) and
\[ \sum_{i=1}^{n} w_i s_i^2 \cdot \Delta_i^2 \leq \|w\|_1. \quad (2.5.11) \]

Moreover \( \frac{1}{n} \|s \odot \Delta\|_2^2 \leq \|s \odot \Delta\|_\infty \leq 1 \). Therefore
\[ \frac{1}{n} \|w\|_1 \sum_{i=1}^{n} s_i^2 \cdot \Delta_i^2 \leq \|w\|_1. \quad (2.5.12) \]

Then the first part of the lemma follows by combining (Equation 2.5.9), (Equation 2.5.10), (Equation 2.5.11), and (Equation 2.5.12). Now since \( r, s \) are nonnegative \( \tilde{w}_i \geq \|w\|_1 \cdot r_i \).
Therefore \( \|\tilde{\Delta}\|_w^2 \geq \|w\|_1 \|\tilde{\Delta}\|_r^2 \), and the second part of the lemma follows from the first part.
By Cauchy-Schwarz on the vectors \( [\sqrt{w_i}]_i \) and \( [\sqrt{\tilde{w}_i} s_i \Delta_i]_i \), definition of \( \tilde{w} \) and first part of the lemma, the third part follows as the following,
\[ \sum_{i=1}^{n} w_i |s_i \Delta_i| \leq \sqrt{\left( \sum_{i=1}^{n} w_i \right) \cdot \left( \sum_{i=1}^{n} w_i s_i^2 \Delta_i^2 \right) \leq \sqrt{\|w\|_1 \cdot \|\Delta\|_w^2} \leq \sqrt{6} \cdot \|w\|_1. \]

Now by definition of \( \tilde{w} \) and first part of the lemma as the following,
\[ \sum_{i=1}^{n} \frac{1}{n} \|w\|_1 \cdot s_i^2 \cdot \Delta_i^2 \leq \|\tilde{\Delta}\|_w^2 \leq 6 \cdot \|w\|_1. \]

Therefore the fourth part follows by
\[ \|s \odot \tilde{\Delta}\|_{\infty} \leq \|s \odot \tilde{\Delta}\|_2 \leq 6n. \]
Algorithm 10: Multiplicative weights update to solve mixed $(2, \infty)$-norm minimization - Part 1 (initialization)

1 **Input:** Full column rank matrix $A \in \mathbb{R}^{n \times (d+1)} (n > d)$, $b \in \mathbb{R}^{d+1}$, $r, s \in \mathbb{R}^n$, $0 < \varepsilon < 1$ such that $\|A^T A\|_F$, $\|(A^T A)^{-1}\|_F$, $\|b\|_2 \leq \kappa$, for $\kappa > 1$.

2 **Output:** $\hat{y} \in \mathbb{R}^n$ such that $\|A\hat{y} - b\|_2 \leq \varepsilon$ and $\|\hat{y}\|_r^2 + \|s \odot \hat{y}\|_\infty \leq \alpha \min_{y : A^T y = b} \|y\|_r^2 + \|s \odot y\|_\infty$.

3 Set $\rho = 8 \cdot n^{1/3} \cdot \log \left( \frac{18\sqrt{6} + 18}{\min_{i \in [n]} (r_i + s_i^2)} \right)$.

4 Set $T = \lceil 2\rho \cdot \log n \rceil$, $t = 0$, $k = 0$, and $w^{(t,0)} \in \mathbb{R}^n$ to be a vector of all ones.

5 For $i \in [n]$, $\eta \in \{0, \ldots, \lceil 16\rho \cdot \log(n) \rceil\}$, set $c_{i,\eta} = 0$.

6 Let $\mu = \|w^{(t,k)}\|_1$ and $p_i = w_i^{(t,k)}$.

7 Set $\tilde{w}_i^{(t,k)} = \tilde{w}_i^{(t,k)} = \mu \cdot r_i + (p_i + \frac{\mu}{n}) \cdot s_i^2$ for all $i \in [n]$.

8 Set DS to be the inverse maintenance data structure // either the dense data structure DDS or the sparse data structure SDS with parameter $m$.

9 **DS.INITIALIZE**($A, w^{(t,k)}$, $\varepsilon$, $\frac{\rho}{10^p(kn)^{30}}$)

For the rest of the proof, we use two potential functions. The first one is $\|w^{(t,k)}\|_1$, which we show only increases slowly over the course of the algorithm. The second potential function is $\min_{\Lambda : A^T \Lambda = b} \|\Lambda\|_{\tilde{w}^{(t,k)}}$ that we show increases significantly when a width reduction steps happen. Moreover, since Lemma 2.5.4.1 guarantees that $\min_{\Lambda : A^T \Lambda = b} \|\Lambda\|_{\tilde{w}^{(t,k)}} = O(1) \cdot \|w^{(t,k)}\|_1$, we get a bound on the maximum number of width reduction steps.

**Lemma 2.5.4.2.** Let $t, k \geq 0$, and $w$ be as defined in Algorithms 10 and 11 (initialized on line 4 of Algorithm 10 and updated on line 13 and line 18 of Algorithm 11). Then we have

$$\|w^{(t+1,k)}\|_1 \leq \left( 1 + \frac{\sqrt{6} + 1}{\rho} \right) \|w^{(t,k)}\|_1,$$

and $\|w^{(t+1,k)}\|_1 \leq \left( 1 + \frac{\sqrt{6} + 1}{\rho} \right) \|w^{(t,k)}\|_1$.

Moreover for $\hat{y} := \frac{1}{T} \sum_{t=1}^{T} y^{(t)}$ and $K$ equal to the number of width reduction steps, we have

$$\|\hat{y}\|_r^2 + \|s \odot \hat{y}\|_\infty \leq 7 \cdot \frac{T + K}{T} + 13.$$
Algorithm 11: Multiplicative weights update to solve mixed \((2, \infty)\)-norm minimization - Part 2 (main loop)

1. **while** \( t < T \) **do**

2.   Set \( \tilde{w}_i^{(t,k)} = \|w_i^{(t,k)}\|_1 \cdot r_i + \left( w_i^{(t,k)} + \frac{1}{n} \|w_i^{(t,k)}\|_1 \right) \cdot s_i^2 \)

3.   **if** \( \|w_i^{(t,k)}\|_1 > 2\mu \) **or** (DS is sparse and \( (n/m)^{1/3} \) divides \( t + k \)) **then**

4.     Set \( S = [n], \mu = \|w_i^{(t,k)}\|_1, p_i = w_i^{(t,k)}, \) and \( \tilde{w}_i^{(t,k)} = \mu \cdot r_i + (p_i + \frac{\mu}{n}) \cdot s_i^2 \)

5.     For \( i \in [n], \eta \in \{0, \ldots, \lceil 16\rho \cdot \log(n) \rceil \}, \) set \( c_{i,\eta} = 0 \)

6.   **else**

7.     Set \( S \leftarrow \{ i \in [n] : c_{i,\eta} \geq \frac{2^\eta}{\log(16\rho \cdot \log(n)) \cdot \log(2)} \} \)

8.     // this guarantees that if \( i \notin S \) then \( w_i^{(t,k)} \leq 2p_i \)

9.     Set \( p_i = w_i^{(t,k)} \) and \( \tilde{w}_i^{(t,k)} = \mu \cdot r_i + (p_i + \frac{\mu}{n}) \cdot s_i^2, \) for all \( i \in S \)

10.    For \( i \in S, \eta \in \{0, \ldots, \lceil 16\rho \cdot \log(n) \rceil \}, \) set \( c_{i,\eta} = 0 \)

11.   **DS.UPDATE** \((S, \tilde{w}_S^{(t,k)})\)

12.   **Let** \( \Delta^{(t,k)} \in \mathbb{R}^n \) **such that** \( \|\pi_a(\Delta^{(t,k)} - \Delta^*(t,k))\|_2 \leq \frac{\epsilon}{16\rho \cdot \log(n)} \|\pi_a\Delta^*(t,k)\|_2 \) **and**

13.   \( \|\Delta^{(t,k)} - \Delta^*(t,k)\|_2 \leq \frac{\epsilon}{2\kappa} \cdot \|\Delta^*(t,k)\|_2 \) **where**

14.   \( \Delta^*(t,k) = \arg\min_{\Delta} \Delta^* \in b \|\Delta\|_2^2 \tilde{w}^{(t,k)} \) // see Lemma 2.1.2.3

15.   **if** \( \|s \odot \Delta^{(t,k)}\|_3 \leq \rho \) **then** // **regular step**

16.     Set \( w_i^{(t+1,k)} = w_i^{(t,k)} \cdot \left( 1 + \frac{\|s \Delta^{(t,k)}\|}{\rho} \right) \) **for all** \( i \in [n] \)

17.     Set \( w_i^{(t+1)} = \Delta^{(t,k)} \)

18.     Increase \( t = t + 1 \)

19.   **else** // **width-reduction step**

20.     Set vector \( u^{(k)} = \frac{\rho}{\|s \odot \Delta^{(t,k)}\|_3} \cdot s \odot \Delta^{(t,k)} \)

21.     Set \( w_i^{(t+1,k)} = w_i^{(t,k)} \cdot \left( 1 + \frac{u_i^{(k)}}{\rho} \right) \) **for all** \( i \in [n] \)

22.     Set \( k = k + 1 \)

23.     **For all** \( i \in [n], \) increase \( c_{i,\eta} \) by one where \( \eta \in \{0, \ldots, \lceil 16\rho \cdot \log(n) \rceil \} \) such that \( \frac{w_i^{(t+1,k)}}{w_i^{(t,k)}} \in (2^{-\eta-1}, 2^{-\eta}], \) where \( w_i^{(t,k)} \) is \( w_i^{(t-1,k)} \) or \( w_i^{(t,k-1)} \) depending on type of the step.

24. **return** \( \frac{1}{T} \sum_{i=1}^{T} y^{(t)} \)

**Proof.** First note that an update to \( w_i^{(t+1,k)} \) happens in a regular step and an update to \( w_i^{(t,k+1)} \) only happens in a width reduction step. However both updates are the same. Therefore we denote either of \( w_i^{(t+1,k)} \) or \( w_i^{(t,k+1)} \) with \( w_i^{(\text{new})} \) in this proof, and prove the bound
for \(w^{(\text{new})}\). Let
\[
\Delta^*(t,k) = \arg\min_{\Delta: \Delta^T\Delta = b} \|\Delta\|^2_{w(t,k)},
\]
and \(\Delta^{(t,k)}\) be as defined on line 11 of Algorithm 11. By triangle inequality
\[
\|w^{(\text{new})}\|_1 = \sum_{i=1}^n w_{i}^{(t,k)} + \frac{1}{\rho} \sum_{i=1}^n w_{i}^{(t,k)} |s_i \cdot \Delta_i^{(t,k)}| \leq \sum_{i=1}^n w_{i}^{(t,k)} + \frac{1}{\rho} \sum_{i=1}^n w_{i}^{(t,k)} |s_i \cdot \Delta_i^{(t,k)}| + \frac{1}{\rho} \sum_{i=1}^n w_{i}^{(t,k)} |s_i \cdot (\Delta_i^{(t,k)} - \Delta_i^{*(t,k)})| = (2.5.13)
\]
\[
(2.5.14)
\]
Therefore by item 3 of Lemma 2.5.4.1,
\[
\|w^{(\text{new})}\|_1 \leq (1 + \frac{\sqrt{6} + 1}{\rho})\|w^{(t,k)}\|_1.
\]
By Lemma 2.1.2.3, we have
\[
\|s \odot (\Delta^{(t,k)} - \Delta^{*(t,k)})\|_\infty \leq \|s \odot (\Delta^{(t,k)} - \Delta^{*(t,k)})\|_2 \leq 2\kappa^6 \varepsilon.
\]
Therefore
\[
\|w^{(\text{new})}\|_1 \leq (1 + \frac{\sqrt{6} + 1}{\rho})\|w^{(t,k)}\|_1.
\]
Taking this over all iterations, denoting the number of width reduction steps with \(K\), and noting that \(\|w^{(0,0)}\| = n\),
\[
\|w^{(T,K)}\|_1 \leq \left(1 + \frac{\sqrt{6} + 1}{\rho}\right)^{T+K} n \leq \exp\left((T + K) \cdot \frac{\sqrt{6} + 1}{\rho}\right) \cdot n. (2.5.15)
\]
Moreover denoting the set of all pairs \((t, k)\), for which \(\Delta^{(t,k)}\) is computed, by \(S\), we have
\[
\begin{align*}
\|w_i^{(T,K)}\|_1 &= \prod_{(t,k) \in S} \left(1 + \frac{|s_i \cdot \Delta_i^{(t,k)}|}{\rho}\right) \geq \prod_{t=1}^T \left(1 + \frac{|s_i \cdot y_i^{(t)}|}{\rho}\right).
\end{align*}
\]
Now note that by construction $|s_i \cdot y_i^{(t)}| \leq \rho$. Therefore

$$w_i^{(T,K)} \geq \prod_{t=1}^{T} \exp \left( \frac{|s_i \cdot y_i^{(t)}|}{2\rho} \right)$$

(2.5.17)

Combining (Equation 2.5.15) and (Equation 2.5.17), taking the logarithm, and using triangle inequality, we have

$$(T + K) \cdot \left( \frac{\sqrt{6} + 1}{\rho} \right) + \log n \geq \sum_{t=1}^{T} \left( \frac{|s_i \cdot y_i^{(t)}|}{2\rho} \right) \geq T \cdot \left( \frac{\frac{1}{T} \sum_{t=1}^{T} y_i^{(t)}}{2\rho} \right) = T \cdot \left( \frac{|s_i \cdot \tilde{y}|}{2\rho} \right)$$

Therefore by definition of $T$ (line 4 of Algorithm 10), we have

$$|s_i \cdot \tilde{y}| \leq 7 \cdot \frac{T + K}{T} + \frac{2\rho}{T} \log(n) \leq 7 \cdot \frac{T + K}{T} + 1$$

Finally by Cauchy-Schwarz inequality

$$\|\tilde{y}\|_r^2 = \sum_{i=1}^{n} r_i \left( \frac{1}{T} \sum_{t=1}^{T} y_i^{(t)} \right)^2 \leq \frac{1}{T} \sum_{i=1}^{n} r_i \sum_{t=1}^{T} (y_i^{(t)})^2 = \frac{1}{T} \sum_{t=1}^{T} \|y^{(t)}\|_r^2$$

This just comes from the convexity of 2-norm and item 2 of Lemma 2.5.4.1. Let $(t, k_i)$ be the pair corresponding to $y^{(t)}$. Then by Lemma 2.1.2.3 and Lemma 2.5.4.1, we have

$$\|y^{(t)}\|_r \leq \|y^{(t)} - \Delta^*(t, k_i)\|_r + \|\Delta^*(t, k_i)\|_r \leq 2\kappa^6 \varepsilon + \|\Delta^*(t, k_i)\|_{\tilde{w}^{(t, k_i)}} \leq 1 + \sqrt{6}$$

Therefore $\|\tilde{y}\|_r^2 \leq 12$. $\square$

Now we use Lemma 2.5.4.1, to prove that the width reduction steps increase the second potential function significantly if the weights have a large increase.

**Lemma 2.5.4.3.** Let $A \in \mathbb{R}^{n \times d}$, $b \in \mathbb{R}^d$, and $w^{(1)}, w^{(2)} \in \mathbb{R}^n_{\geq 0}$ such that $w^{(2)} - w^{(1)} \geq 0$. 

121
Moreover for \( j = 1, 2 \), let
\[
\Delta^{(j)} = \arg \min_{\Delta : \Delta^T \Delta = b} \frac{1}{2} \| \Delta \|_{w(j)}^2.
\]

Then
\[
\frac{1}{2} \| \Delta^{(2)} \|_{w(2)}^2 \geq \frac{1}{2} \| \Delta^{(1)} \|_{w(1)}^2 + \frac{1}{4} \| \Delta^{(1)} \|_{w(2), w(1)}^2.
\]

**Proof.** Throughout the proof let \( j \in \{1, 2\} \). We look at the dual of
\[
\min_{A^T \Delta = b} \frac{1}{2} \| \Delta \|_{w(j)}^2,
\]
which is
\[
\max_z -b^T z - \frac{1}{2} \|Az\|_{w(j)-1}^2.
\]

Let \( z^{(j)} \) be the optimal solution of the dual problem for \( w^{(j)} \). The gradient of \( \| \Delta \|_{w(j)}^2 \) at \( \Delta^{(j)} \) is orthogonal to the kernel of \( A^T \). Therefore there exists \( u^{(j)} \in \mathbb{R}^d \) such that
\[
W^{(j)} \Delta^{(j)} = Au^{(j)}.
\]

Therefore \( \Delta^{(j)} = (W^{(j)})^{-1} Au^{(j)} \), and since \( A^T \Delta^{(j)} = b \), we have
\[
u^{(j)} = (A^T (W^{(j)})^{-1} A)^{-1} b, \quad \text{and} \quad \Delta^{(j)} = (W^{(j)})^{-1} A (A^T (W^{(j)})^{-1} A)^{-1} b
\]

For the dual solution, we have that the gradient of \( -b^T z - \frac{1}{2} \|Az\|_{w(j)-1}^2 \) at \( z^{(j)} \) is zero. Therefore
\[
-b - A^T (W^{(j)})^{-1} Az^{(j)} = 0.
\]

Therefore \( z^{(j)} = -(A^T (W^{(j)})^{-1} A)^{-1} b \), and
\[
(Az^{(j)})^2 = \left( A (A^T (W^{(j)})^{-1} A)^{-1} b \right)_i^2, \quad \text{and} \quad (\Delta^{(j)})^2 = \left( w^{(j)}_i \right)^2 \left( A (A^T (W^{(j)})^{-1} A)^{-1} b \right)_i^2.
\]
Thus,

\[(w_i^{(j)})^{-1} (Az_i^{(j)})_i^2 = w_i^{(j)} (\Delta_i^{(j)})^2, \text{ and } \|\Delta_i^{(j)}\|_{W^{(j)}}^2 = \|Az_i^{(j)}\|_{W^{(j)}}^2 = b^T (A^T W^{(j)})^{-1} A^{-1} b,\]

(2.5.18)

where the last equality follows by substituting the value of \(z^{(j)}\). We have

\[-\frac{1}{w_i^{(2)}} = -\frac{1}{w_i^{(1)}} + \frac{w_i^{(2)} - w_i^{(1)}}{w_i^{(2)} \cdot w_i^{(1)}}.

Therefore,

\[
\frac{1}{2} \|\Delta_i^{(2)}\|_{W^{(2)}}^2 = -b^T z_i^{(2)} - \frac{1}{2} \|Az_i^{(2)}\|_{(W^{(2)})^{-1}}^2 \geq -b^T z_i^{(1)} - \frac{1}{2} \|Az_i^{(1)}\|_{(W^{(2)})^{-1}}^2 = -b^T z_i^{(1)} - \frac{1}{2} \sum_{i \in [n]} (w_i^{(2)})^{-1} (Az_i^{(1)})_i^2

= -b^T z_i^{(1)} - \frac{1}{2} \sum_{i \in [n]} (w_i^{(1)})^{-1} (Az_i^{(1)})_i^2 + \frac{1}{2} \sum_{i \in [n]} \frac{(w_i^{(2)} - w_i^{(1)})}{w_i^{(2)} \cdot w_i^{(1)}} (w_i^{(2)})^{-1} (Az_i^{(1)})_i^2

= \frac{1}{2} \|\Delta_i^{(1)}\|_{W^{(1)}}^2 + \frac{1}{2} \sum_{i \in [n]} \frac{(w_i^{(2)} - w_i^{(1)})}{w_i^{(2)} \cdot w_i^{(1)}} \cdot (w_i^{(1)}) (Az_i^{(1)})_i^2.
\]

We are now equipped to show that in a width reduction step, the value of the second potential function increases significantly. This combined with Lemma 2.5.4.2 implies that the number of width reduction steps is at most \(\tilde{O}(\rho)\).

**Lemma 2.5.4.4.** Let

\[
\Delta^{\ast (r,k)} = \arg \min_{\Delta: A^T \Delta = b} \frac{1}{2} \|\Delta\|_{W^{(r,k)}}^2,
\]
where $\overline{W}^{(t,k)}$ is defined as line 2 of Algorithm 11. Then after a width-reduction step

$$\frac{1}{2} \left\| \Delta^{*}(t,k) \right\|_{\overline{W}(t,k+1)}^2 \geq \left( 1 + \frac{\rho^2}{72 \cdot n} \right) \cdot \frac{1}{2} \left\| \Delta^{*}(t,k) \right\|_{\overline{W}(t,k)}^2,$$

where $\rho$ is defined as line 3 of Algorithm 10.

Proof. First note that for positive numbers $a, b, c$ and $r \geq 1$ with $b \geq a$,

$$\frac{a + c}{b + rc} \geq \frac{1}{r} \cdot \frac{a}{b}.$$

This holds by dividing both sides of the following by $r$

$$\frac{ra + rc}{b + rc} \geq \frac{ra}{b} \geq \frac{a}{b}.$$

By Lemma 2.5.4.3, in the width reduction step,

$$\frac{1}{2} \left\| \Delta^{*}(t,k+1) \right\|_{\overline{W}(t,k+1)}^2 \geq \frac{1}{2} \left\| \Delta^{*}(t,k) \right\|_{\overline{W}(t,k)}^2 + \frac{1}{2} \sum_{i \in [n]} \frac{\overline{w}_i^{(t,k+1)} - \overline{w}_i^{(t,k)}}{\overline{w}_i^{(t,k)}} \cdot \overline{w}_i^{(t,k)} (\Delta^{*}_i)^2.$$

Now by construction and Lemma 2.5.4.2,

$$\left\| w^{(t,k)} \right\|_1 \leq \left\| w^{(t,k+1)} \right\|_1 \leq (1 + \frac{12}{\rho}) \left\| w^{(t,k)} \right\|_1 \leq 2 \left\| w^{(t,k)} \right\|_1,$$

and since $w_i^{(t,k+1)} \geq 1$ and $\left\| w^{(t,k+1)} \right\|_1 \leq 6n$ over the course of the algorithm,

$$w_i^{(t,k+1)} + \frac{1}{n} \left\| w^{(t,k+1)} \right\|_1 \leq w_i^{(t,k+1)} + 6 = (1 + \frac{u_i^{(k)}}{\rho}) \cdot w_i^{(t,k)} + 6w_i^{(t,k)} \leq 8 \cdot w_i^{(t,k)},$$

124
where the last inequality follows from the construction of $\mathbf{u}^{(k)}$. Then by (Equation 2.5.19),

$$
\frac{\mathbf{w}^{(t,k)}}{\mathbf{w}^{(t,k+1)}} = \left[ \|\mathbf{w}^{(t,k)}\|_1 \cdot \mathbf{r}_i + \left( \|\mathbf{w}^{(t,k)}\|_1 + \frac{1}{n} \|\mathbf{w}^{(t,k+1)}\|_1 \right) \right] \cdot \mathbf{s}_i^2
$$

Moreover since $\|\mathbf{w}^{(t,k)}\|_1 \leq \|\mathbf{w}^{(t,k+1)}\|_1$,

$$
\mathbf{w}^{(t,k+1)} - \mathbf{w}^{(t,k)} \geq (\mathbf{w}^{(t,k+1)} - \mathbf{w}^{(t,k)}) \cdot \mathbf{s}_i^2.
$$

Therefore

$$
\frac{1}{2} \left\| \mathbf{A}^{+(t,k+1)} \right\|_2^2 \geq \frac{1}{2} \left\| \mathbf{A}^{+(t,k)} \right\|_2^2 + \frac{1}{24} \sum_{i \in [n]} \frac{(\mathbf{w}^{(t,k+1)} - \mathbf{w}^{(t,k)}) \cdot \mathbf{s}_i^2}{\|\mathbf{w}^{(t,k)}\|_1} \cdot \frac{\|\mathbf{w}^{(t,k)}\|_1 \cdot \mathbf{s}_i^2 (\mathbf{A}^{+(t,k)})^2}{\mathbf{r}_i}.
$$

Now by construction of $\mathbf{u}^{(k)}$ (since $\|\mathbf{s} \otimes \mathbf{A}^{+(t,k)}\|_3 > \rho$), and construction of $\mathbf{w}^{(t,k+1)}$, we have

$$
\frac{1}{2} \left\| \mathbf{A}^{+(t,k+1)} \right\|_2^2 \geq \frac{1}{2} \left\| \mathbf{A}^{+(t,k)} \right\|_2^2 + \frac{\|\mathbf{w}^{(t,k)}\|_1}{24 \cdot n} \sum_{i \in [n]} \frac{\mathbf{u}_i^{(k)}}{\mathbf{r}_i} \cdot (\mathbf{u}_i^{(k)})^2
$$

Finally by Lemma 2.5.4.1, $\left\| \mathbf{A}^{+(t,k)} \right\|_2^2 \leq 6 \cdot \|\mathbf{w}^{(t,k)}\|_1$. Thus,

$$
\frac{1}{2} \left\| \mathbf{A}^{+(t,k+1)} \right\|_2^2 \geq \left( 1 + \frac{\rho^2}{72 \cdot n} \right) \cdot \frac{1}{2} \left\| \mathbf{A}^{+(t,k)} \right\|_2^2.
$$
We finally bound the number of changes to \( \hat{w} \). This is the main factor in the running time of the inverse maintenance procedure.

**Lemma 2.5.4.5.** Let \( \hat{T} := T + K \) be the number of iterations of Algorithm 11, and \( k \in [\hat{T}] \). For \( t \in \hat{T} \) and \( \eta \in \{0, \ldots, \left\lceil \log_2(\hat{T}) \right\rceil \) \( c_{t,\eta} \) be the number of entries of \( w \) that change by a factor in the interval of \((2^{-\eta-1}, 2^{-\eta}]\). Then

\[
\sum_{t=1}^{\hat{T}} c_{t,\eta} \leq 2^{3(\eta+1)}.
\]

**Proof.** Note that in a regular step, the relative change (i.e., \((w_i^{(t+1,k)} - w_i^{(t,k)})/w_i^{(t,k)}\)) to each entry is at most \(|s_i \cdot \Delta_i^{(t,k)}|/\rho < 1\). Moreover by the upper bound of \(\|s \odot \Delta^{(t,k)}\|_3\) in the regular steps and the construction of \(u^{(k)}\) in the width-reduction steps, we have that

\[
\left\| \frac{w^{(\text{new})} - w^{(t,k)}}{w^{(t,k)}} \right\|_3 \leq 1, \tag{2.5.20}
\]

where \(w^{(\text{new})}\) is either \(w^{(t+1,k)}\) or \(w^{(t,k+1)}\) depending on the type of the step. Therefore the number of changes of factor in \((2^{-\eta-1}, 2^{-\eta}]\) in one step is at most \(2^{3\eta+3}\) and the number of such changes over the course of the algorithm is \(\hat{T} \cdot 2^{3\eta+3}\). \(\Box\)

**Proof of Theorem 2.1.1.9.** We show that Algorithms 10 and 11 achieve the desired result if

\[
0.5 \leq \min_{\Delta: A^\top \Delta = b} \|\Delta\|_F^2 + \|s \odot \Delta\|_\infty < 1. \tag{2.5.21}
\]

We require this assumption to be able to use the results we developed in this section, e.g., Lemma 2.5.4.1. Note that if we scale all of \(r\) and \(s\) by a number \(\alpha\), the minimum value is also scaled by \(\alpha\). Therefore we only need to “guess” the correct scaling factor as a power of two. This means that we try to minimize the objective function with different scaling factors and then we take the minimum over the vectors return for these different scaling
factors. Note that this only affects the running time of the algorithm. Later in the proof, when we discuss the running time, we take the number of scaling factors we need to try into consideration.

We first need to bound the number of iterations. The number of regular iterations is bounded by \( \lceil 2\rho \cdot \log n \rceil \) by construction. Let \( K \) be the number of width-reduction steps of the algorithm. Then since \( w^{(0,0)} = \hat{1} \), by Lemma 2.5.4.2 for all steps \((t, k)\) of the algorithm,

\[
\|w^{(t,k)}\|_1 \leq \exp \left( \frac{\sqrt{6} + 1}{\rho} \cdot (T + K) \right) \cdot n \leq \exp \left( \frac{\sqrt{6} + 1}{\rho} \cdot K \right) \cdot (3\sqrt{6} + 3) \cdot n^2 \quad (2.5.22)
\]

Now let \( \bar{\Delta}^{(t,k)} := \arg \min_{\Delta: A^\top \Delta = b} ||\Delta||_2^2 \). Since \( w^{(t,k)} = \hat{1} \) and \( \bar{w}^{(t,k)} = \|w^{(t,k)}\|_1 \cdot r_i + \left( w^{(t,k)}_i + \frac{1}{n} \|w^{(t,k)}\|_1 \right) \cdot s_i^2 \), defining \( u \in \mathbb{R}^n \) as \( u = \hat{1} \cdot \min_{i \in [n]} \{ n \cdot r_i + 2 \cdot s_i^2 \} \), we have

\[
u \leq \bar{w}^{(0,0)}.
\]

Therefore by Lemma 2.5.4.3,

\[
\left\| \bar{\Delta}^{(0,0)} \right\|_2^2 \geq \min_{i \in [n]} \{ r_i + s_i^2 \} \cdot \min_{\Delta: A^\top \Delta = b} ||\Delta||_2^2
\]

Since for a linear system, product of the pseudoinverse and the vector gives the solution with minimum 2-norm and \( A \) has full column rank,

\[
\arg \min_{\Delta: A^\top \Delta = b} \|\Delta\|_2^2 = A(A^\top A)^{-1} A^\top b.
\]

Since by Lemma 2.5.4.4 for each width reduction step,

\[
\left\| \bar{\Delta}^{(t,k+1)} \right\|_2^2 \geq \left( 1 + \frac{\rho^2}{72 \cdot n} \right) \cdot \left\| \bar{\Delta}^{(t,k)} \right\|_2^2
\]
we have 
\[ \| \hat{\Delta}^{(T,K)} \|_{\| \hat{w}^{(T,K)} \|}^2 \geq \exp(\frac{K \cdot \rho^2}{144 \cdot n} \cdot \| \pi_A b \|_2^2 \cdot \min_{i \in [n]} \{ r_i + s_i^2 \}) \]

Moreover by (Equation 2.5.22) and Lemma 2.5.4.1,
\[ \| \hat{\Delta}^{(T,K)} \|_{\| \hat{w}^{(T,K)} \|}^2 \leq 6 \cdot \| w^{(t,k)} \|_1 \leq \exp \left( \frac{\sqrt{6} + 1}{\rho} \cdot K \right) \cdot \left( 18\sqrt{6} + 18 \right)^n. \]

Therefore 
\[ \frac{K \cdot \rho^2}{144 \cdot n} + \log \left( \| \pi_A b \|_2^2 \cdot \min_{i \in [n]} \{ r_i + s_i^2 \} \right) \leq \frac{\sqrt{6} + 1}{\rho} \cdot K + \log \left( \left( 18\sqrt{6} + 18 \right)^n \right). \]

Therefore 
\[ K \leq \frac{144 \cdot n \rho}{\rho^3 - 144 \cdot (\sqrt{6} + 1)n} \log \left( \frac{\left( 18\sqrt{6} + 18 \right)^n}{\| \pi_A b \|_2^2 \cdot \min_{i \in [n]} \{ r_i + s_i^2 \}} \right), \tag{2.5.23} \]

Since $144 \cdot (\sqrt{6} + 1) < 500$ and $\rho \geq 8 \cdot n^{1/3}$, $\rho^3 - 144 \cdot (\sqrt{6} + 1)n$ is positive, and
\[ \frac{144 \cdot n \rho}{\rho^3 - 144 \cdot (\sqrt{6} + 1)n} \leq \frac{144 \cdot n \rho}{12 \cdot n} = 96n^{1/3}. \tag{2.5.24} \]

Therefore $K$ and the number of iterations of the algorithm are
\[ K = \tilde{O} \left( n^{1/3} \log \left( \frac{1}{\| \pi_A b \|_2^2 \cdot \min_{i \in [n]} \{ r_i + s_i^2 \}} \right) \right), \tag{2.5.25} \]

which by Remark 2.1.4.2 (since if $\| \pi_A b \|_2$ is too small, we can return the vector of all zeros as the solution) is
\[ K = \tilde{O} \left( n^{1/3} \log \left( \frac{\kappa}{\min_{i \in [n]} \{ r_i + s_i^2 \}} \right) \right). \tag{2.5.26} \]

Therefore by (Equation 2.5.23), (Equation 2.5.24), and Lemma 2.5.4.2, for the output of
the algorithm $\hat{y} := \frac{1}{T} \sum_{t=1}^{T} y^{(t)}$ we have

$$\|\hat{y}\|_r^2 + \|s \circ \hat{y}\|_\infty \leq 7 \cdot (1 + 6) + 13 \leq 62.$$  

Since by (Equation 2.5.21), the optimal objective value is at least a half, this implies that we achieve a constant factor approximation. Note that for all $\Delta^{*\star(t,k)}$, and $x^*$,

$$\pi_A x^* = A(A^\top A)^{-1} A^\top x^* = A(A^\top A)^{-1} b = A(A^\top A)^{-1} A^\top \Delta^{*\star(t,k)} = \pi_A \Delta^{*\star(t,k)}$$

Therefore since for all $(t, k)$,

$$\left\| \pi_A (\Delta^{(t,k)} - \Delta^{*\star(t,k)}) \right\| \leq \varepsilon \left\| \pi_A \Delta^{*\star(t,k)} \right\|,$$

for all $t \in [T]$,

$$\left\| \pi_A (y^{(t)} - x^*) \right\| \leq \varepsilon \left\| \pi_A x^* \right\|.$$  

Thus by triangle inequality,

$$\left\| \pi_A (\hat{y} - x^*) \right\| \leq \varepsilon \left\| \pi_A x^* \right\|.$$  

Finally, we need to bound the running time. The number of different scaling factors we need to try to guarantee (Equation 2.5.21). Note that for any $\Delta \in \mathbb{R}^n$,

$$r_{\min} \cdot \|\Delta\|_r^2 \leq \|\Delta\|_r^2 + \|s \circ \Delta\|_\infty \leq 2 \cdot \max\{r_{\max}, s_{\max}\} \cdot \max\{\|\Delta\|_2^2, \|\Delta\|_2\},$$

where $r_{\min} = \min_{i\in[n]} r_i$, $r_{\max} = \max_{i\in[n]} r_i$, and $s_{\max} \max_{i\in[n]} s_i$. Therefore

$$r_{\min} \cdot \|\pi_A x^*\|_r^2 \leq \min_{\Delta=b} \|\Delta\|_r^2 + \|s \circ \Delta\|_\infty \leq 2 \cdot \max\{r_{\max}, s_{\max}\} \cdot \max\{\|\pi_A x^*\|_2^2, \|\pi_A x^*\|_2\}.$$  

Therefore the number of scaling factors we need to try to have the guarantee that for one
of them (Equation 2.5.21) holds is at most

\[\log(2(r_{\text{max}} + s_{\text{max}}) \max\{1, 1/\|\pi A^*\|_2\}/r_{\text{min}}) = O(\log(\kappa(r_{\text{max}} + s_{\text{max}})/(r_{\text{min}} \cdot \epsilon))).\]

where the equality follows from Remark 2.1.4.2. We now bound the running time of Algorithms 10 and 11 in the dense case. We first bound the running time of inverse maintenance. Note that the inverse is either updated through line 4 or line 8 of the algorithm. The former is triggered when the 1-norm of the weights is changed by a factor of two, which only occurs \(O(\log n)\) times by (Equation 2.5.22), (Equation 2.5.25), and because \(w^{(0,0)} = 1\). Therefore the cost of such updates is bounded by \(\tilde{O}(n^\omega \log(k/\epsilon))\). Now consider updates through line 8 of the algorithm. For an index \(i \in [n]\) suppose the entry \(i\) of \(\hat{w}\) has changed in iterations \(s\) and \(e\) and has been fixed between these two iterations. Moreover, suppose \(1 + q_t\) be the relative change of entry \(i\) of \(w\) at step \(t\). Since an entry of \(\hat{w}\) changes only when the corresponding entry of \(w\) has changed by more than a factor of two, we have

\[\exp(\sum_{t=e}^{s-1} q_t) \geq \prod_{t=e}^{s-1} (1 + q_t) \geq 2.\]

Now if for all \(\eta \in \{0, \ldots, \left\lfloor \log_2(T) \right\rfloor\}\), where \(\hat{T} := T + K\) is the number of iterations of the algorithm, the number of \(q_t\)’s for \(t \in \{e, \ldots, s - 1\}\) is less than \(\frac{2^\eta}{\log_2(\hat{T}) \log(2)}\), then

\[\exp(\sum_{t=e}^{s-1} q_t) < 2.\]

Therefore for at least one of the \(\eta\)’s, the number of such \(q_t\)’s is at least

\[\frac{2^\eta}{\log_2(\hat{T}) \log(2)}.\]

Therefore by Lemma 2.5.4.5, the sum of the rank of the updates caused by changes between \((2^{-\eta-1}, 2^\eta]\) through line 8 of the algorithm is at most

\[\hat{T}2^{2\eta+3} \cdot \left\lfloor \log_2(\hat{T}) \right\rfloor \cdot \log(2) = \tilde{O}\left(\hat{T}2^{2\eta}\right).\]  

(2.5.27)

By concavity of \((\cdot)^{\omega-2}\) and since we only add entries that have changed due to accumula-
tions of changes in \((2^{-\eta} - 1, 2^\eta]\) once every \(2^\eta\) iterations, the cost of such updates is

\[
\tilde{O} \left( \frac{\widetilde{T}}{2^n} n^2 \left( \frac{\widetilde{T} 2^{2\eta}}{T/2^n} \right)^{\omega-2} \right) = \tilde{O} \left( \widetilde{T} \cdot n^2 \cdot 2^{\eta(3(\omega-2)-1)} \cdot \log(\kappa/\epsilon) \right).
\]

Since \(3(\omega - 2) - 1 > 0\) for the current value of \(\omega\), this is increasing in \(\eta\), and therefore the total cost for updates through line 8 of the algorithm is

\[
\tilde{O} \left( \widetilde{T}^{3(\omega-2)} \cdot n^2 \cdot \log(\widetilde{T}) \cdot \log(\kappa/\epsilon) \right),
\]

which by (Equation 2.5.26) and definition of \(T\) is

\[
\tilde{O} \left( n^{\omega} \log \left( \frac{\kappa}{\min_{i \in [n]} \{ r_i + s_i^2 \}} \right) \cdot \log(\kappa/\epsilon) \right).
\]

By Lemma 2.1.2.3, the overall cost of solving the constrained weighted linear regression problems is

\[
\tilde{O} \left( \widetilde{T} \cdot n^2 \log(\kappa) \log^2(1/\epsilon) \right) = \tilde{O} \left( n^{7/3} \log \left( \frac{\kappa}{\min_{i \in [n]} \{ r_i + s_i^2 \}} \right) \cdot \log(\kappa) \log^2(1/\epsilon) \right).
\]

We now consider the sparse case. First, note that the only randomization comes from the construction and reconstruction of the sparse inverse. Taking the union bound and upper bounding the number of reconstructions by the total number of iterations of the algorithm gives the probability bound. We now bound the running time. First note that the number of reconstructions of the sparse inverse triggered by line 4 of the algorithm because the 1-norm of \(w\) has changed by a factor of two is only \(O(\log n)\) as discussed above. Moreover line 4 is triggered once every \((n/m)^{1/3}\) iterations. Therefore the total cost of line 4 is

\[
\tilde{O} \left( \text{NNZ}(A) \cdot m \cdot n + n^\omega m^{2-\omega} \log^2(\kappa/\epsilon) \cdot m^{1/3} \log \left( \frac{\kappa}{\min_{i \in [n]} \{ r_i + s_i^2 \}} \right) \right),
\]

131
which is

\[ \tilde{O} \left( \left( \text{NNZ}(A) \cdot m^{4/3} \cdot n + n^\omega m^{7/3-\omega} \right) \log^2(\kappa/\epsilon) \cdot \log(\alpha_1 \kappa) \right). \]

Now note that the only \( \eta \) that can cause an index to be added to the set \( S \) through line 8 are the ones with

\[ \frac{2^n}{\log_2(16 \rho \log n) \log(2)} \leq (n/m)^{1/3}. \]

Otherwise, the changes are too small to accumulate enough in \((n/m)^{1/3}\) iterations before a total reconstruction of the sparse inverse through line 4 is triggered. Now consider reconstructions of the inverse triggered by line 8. For one \( \eta \), by our above bounds on the number of changes (Equation 2.5.27) is \( \tilde{T}^{2^{n+3}} \).

Therefore the cost of such reconstruction is

\[ \tilde{O} \left( \left( \text{NNZ}(A) \cdot m \cdot n + n^\omega m^{2-\omega} \right) \log^2(\kappa/\epsilon) \cdot \frac{\tilde{T}^{2^n}}{n/m} \right). \]

Since this is increasing in \( \eta \), taking the large possible \( \eta \) and replacing \( \tilde{T} \) by its value, this is

\[ \tilde{O} \left( \left( \text{NNZ}(A) \cdot m \cdot n + n^\omega m^{2-\omega} \right) \log^2(\kappa/\epsilon) \cdot \frac{n^{1/3} \log(\alpha_1 \kappa) (n/m)^{2/3}}{n/m} \right), \]

which is

\[ \tilde{O} \left( \left( \text{NNZ}(A) \cdot m^{4/3} \cdot n + n^\omega m^{7/3-\omega} \right) \log^2(\kappa/\epsilon) \cdot \log(\alpha_1 \kappa) \right). \]

We now bound the cost of updates to the inverse through the Woodbury identity. In this case, by Theorem 2.3.2.2, and since \((\cdot)^{\omega-2}\) is a concave function, for any \( \eta \), the cost is

\[ \tilde{O} \left( \left( \text{NNZ}(A) \cdot m^2 \cdot \tilde{T}^{2\eta} + n^2 \cdot \frac{\tilde{T}}{2^n} \cdot \left( \frac{\tilde{T}^{2\eta}}{\tilde{T}/2^n} \right)^{\omega-2} \right) \log^2(\kappa/\epsilon) \right). \]

Since we only need to consider \( \eta \) such that \[ \frac{2^n}{\log_2(16 \rho \log n) \log(2)} \leq (n/m)^{1/3}, \]
and this is increasing in \( \eta \), the total cost of these updates is

\[ \tilde{O} \left( \left( \text{NNZ}(A) \cdot m^{4/3} \cdot n + n^\omega m^{7/3-\omega} \right) \log^2(\kappa/\epsilon) \cdot \log(\alpha_1 \kappa) \right). \]
Finally, by Theorem 2.3.2.2 and Lemma 2.1.2.3, the cost of solving constrained weighted regression problems is

\[ \tilde{O} \left( \left( \text{NNZ}(A) \cdot m^2 + n^2 \right) \log^2(\kappa/\epsilon) \cdot T \right), \]

which since \( m < n \) is

\[ \tilde{O} \left( \left( \text{NNZ}(A) \cdot m^{4/3} \cdot n + n^\omega m^{7/3-\omega} \right) \log^2(\kappa/\epsilon) \cdot \log(\alpha_1 \kappa) \log^2(1/\epsilon) \right). \]

Combining these with the number of scaling factors we need to try to guarantee that for one of them (Equation 2.5.21) holds gives the total running time. □

2.5.5 Mixed \((2, p)\)-Norm Minimization

In this section, we consider the bit complexity of solving the mixed \((2, p)\)-norm minimization (Equation 2.5.4) directly. Similar to the mixed \((2, \infty)\)-norm minimization, we utilize the multiplicative weights update algorithm, width reduction, and inverse maintenance techniques. The main theorem of this section is the following, which can also be improved beyond the (current) matrix multiplication time for sparse matrices by the data structure of Theorem 2.3.2.2.

**Theorem 2.5.5.1.** Let \( p > 1, 1 > \epsilon > 0, z \in \mathbb{R}, A \in \mathbb{R}^{n \times d}, g \in \mathbb{R}^n, \) and \( t \in \mathbb{R}^n_{\geq 0} \) such that \( n^{-1/p} \leq t_i \leq 1, \) for all \( i \in [n] \). Moreover, suppose \( \|A^T A\|_F, \| (A^T A)^{-1} \|_F \leq \kappa, \) and

\[ \| (I - A (A^T A)^{-1} A^T) g \|_2 \geq \frac{\epsilon}{\kappa}. \]

Moreover, suppose the optimal value of the following problem is at most one.

\[
\min_{\Delta \in \mathbb{R}^n} \quad \gamma_p(t, \Delta) \tag{2.5.28}
\]

s.t. \( g^T \Delta = z, \)
\[ \mathbf{A} \mathbf{A} = 0. \]

Then there exists an algorithm that computes a constant factor approximation to this problem in time \( \tilde{O}_p((n^\omega + n^{7/3}) \cdot \log(\kappa/\varepsilon)) \).

**Algorithm 12:** Algorithm for the adjusted mixed \((2, p)\)-norm problem - Part 1 (initialization)

1. **Input:** \( \mathbf{A} \in \mathbb{R}^{n \times d}, \mathbf{t} \in \mathbb{R}^n, \mathbf{g} \in \mathbb{R}^n, z \in \mathbb{R}, p \in (1, \infty) \)
2. Set \( \overline{\mathbf{A}} = [\mathbf{A} | \mathbf{g}] \) and \( \overline{\mathbf{b}} = [\overline{\mathbf{0}}_d | z] \)
3. Set \( \rho = \tilde{\Theta}_p(n^{(p^2-4p+2)/(3p-2)}), \beta = \tilde{\Theta}_p(n^{p-2/(3p-2)}), \)
   \[ \alpha = \tilde{\Theta}_p \left(n^{-\frac{(p^2-5p+2)}{3p-2}} \left(\log(n \left\| \mathbf{A} \right\|_2^2 / \left\| \mathbf{b} \right\|_2^2) \right)^{\frac{-p}{1-p}}\right), \]
   and
   \[ \tau = \tilde{\Theta}_p \left(n^{-\frac{(p-1)(p-2)}{3p-2}} \left(\log(n \left\| \mathbf{A} \right\|_2^2 / \left\| \mathbf{b} \right\|_2^2) \right)^{\frac{p(p-1)}{3p-2}}\right) \]
   // \( \rho \) is width parameter, \( \beta \) is threshold for \( r \), \( \alpha \) is step size, \( \tau \) is threshold for \( p \)-norm. The constants are picked so the relations in Lemmas 2.5.5.5 and 2.5.5.7 are satisfied.
4. Set \( T = \alpha^{-1}n^{1/p}, i = k = 0, w^{(i,k)} = \overline{\mathbf{0}} \) and \( \mathbf{x} = \overline{\mathbf{0}} \)
5. \( \tilde{r}^{(i,k)}_{j} \leftarrow (n^{1/p}t_{j})^{p-2}, \forall j \in [n] \)
6. Set DS to be the inverse maintenance data structure // either the dense data structure DDS or the sparse data structure SDS with parameter \( m \)
7. DS.INITIALIZE(\( \overline{\mathbf{A}}, \tilde{r}^{(i,k)}, \frac{\varepsilon}{\tilde{O}(\kappa \cdot m^{30})} \))

We start by adjusting the vector \( \mathbf{t} \) and the number \( z \), so that all of the entries of \( \mathbf{t} \) are within a polynomial (in \( n \)) bound, the corresponding problem has an optimal value less than or equal to one, and an approximate solution to the adjusted problem gives an approximate solution to the original mixed \((2, p)\)-norm problem.

**Lemma 2.5.5.2.** Let \( p \geq 2, \mathbf{t}, \mathbf{g} \in \mathbb{R}^n, \mathbf{t} \geq 0, \) and \( j \in \mathbb{Z} \) such that the following is feasible
Algorithm 13: Algorithm for the adjusted mixed $(2, p)$-norm problem - Part 2
(main loop)

while $i < T$ do

(1) Find the significant buckets and update the preconditioner.

$r(i,k) \leftarrow (n^{1/p}t)^{p-2} + (w(i,k))^{p-2}$

For all $j \in [n]$ find the least non-negative integer $\eta_j$ such that
\[
\frac{1}{2^{\eta_j}} \leq \frac{r(i,k)_j - r^{(old)}_j}{r_j}
\]

$r^{(old)} \leftarrow r(i,k)$

For all $j \in [n], c_j, \eta_j \leftarrow c_j, \eta_j + 1$

if DS is sparse and $\left\lfloor n^{(p-2)/(3p-2)}/m^{1/3} \right\rfloor$ divides $i$ then

$S \leftarrow [n]$

else

$S \leftarrow \bigcup_{\eta;i+1 \mod 2^{\eta} \equiv 0} \{j : c_j, \eta_j \geq 2^{\eta}\}$

$\bar{r}(i,k)_j \leftarrow r(i,k)_j, \forall j \in S$

$c_j, \eta_j \leftarrow 0$ for all $(j, \eta)$ such that $j \in S$.

DS.UPDATE($S, \bar{r}(i,k)$)

(2) Solve the weighted linear regression by Richardson’s iteration and
preconditioning (Lemma 2.1.2.3).

Let $\Delta(i,k) \in \mathbb{R}^n$ such that $\|\pi_{\bar{X}}(\Delta(i,k) - \Delta^*(i,k))\|_2 \leq \frac{\varepsilon}{T} \|\pi_{\bar{X}}\Delta^*(i,k)\|_2$ and
\[
\|\Delta(i,k) - \Delta^*(i,k)\|_2 \leq \frac{\varepsilon}{2}\|\Delta^*(i,k)\|_{r(i,k)}
\]
where

$\Delta^*(i,k) = \arg\min_{\Delta, \bar{X}^{(i,k)}} \|\Delta\|_{r(i,k)}$

(3) Update the weights.

if $\|\Delta\|_p \leq \tau$ then // regular step

$w_{j}^{(i+1,k)} \leftarrow w_{j}^{(i,k)} + \alpha |\bar{X}_j|, \forall j \in [n]$

$x \leftarrow x + \alpha \bar{X}$

Set $i = i + 1$

else // width-reduction step

For all $j \in [n]$ with $|\bar{X}_j| \geq \rho$ and $r_j \leq \beta$, set
\[
w_{j}^{(i,k+1)} = 4^{1/(p-2)} \max\{n^{1/p}t_j, w_{j}^{(i,k)}\}.
\]

For rest of $j \in [n]$, set $w_{j}^{(i,k+1)} = w_{j}^{(i,k)}$.

Set $k = k + 1$

return $n^{-1/p}x$

for some $\Delta \in \mathbb{R}^n$. 

\[
\gamma_p(t, \Delta) \leq \frac{p}{p - 1}2^{j+p},
\] 

(2.5.29)
\[ \mathbf{g}^\top \mathbf{\Delta} = 2^j, \]
\[ \mathbf{A}^\top \mathbf{\Delta} = 0. \]

Moreover for all \( i \in [n] \), let \( \widehat{z}_i = \left( \frac{2}{p} \right)^{1/2} \left( \frac{p-1}{p} \right)^{1/p} 2^{j(1-1/p)-2} \)
\[ \widehat{t}_i = \min \left\{ \max \left\{ \left( \frac{p-1}{p} \right)^{1/p} \frac{1}{2^{1+j/p} t_i}, n^{-1/p} \right\}, 1 \right\}. \]

Also let
\[ \mathbf{\Delta}^* = \arg \min_{\mathbf{\Delta}: \mathbf{A}^\top \mathbf{\Delta} = \mathbf{0}, \mathbf{g}^\top \mathbf{\Delta} = \widehat{z}} \gamma_p(\widehat{t}, \mathbf{\Delta}) \leq 1. \]

Then \( \gamma_p(\widehat{t}, \mathbf{\Delta}^*) \leq 1 \), and for \( \widehat{t} \in \mathbb{R}^n \) such that \( \gamma_p(\widehat{t}, \mathbf{\Delta}) \leq \beta \),
\[ \gamma_p(\widehat{t}, \mathbf{\Delta}) \leq \left( \frac{p}{2} \right)^{p/2} \cdot \frac{p \cdot 2^{p+j}}{p-1} \cdot (\beta + 1), \]
where \( \mathbf{\Delta} = \left( \frac{p}{2} \right)^{1/2} \cdot \left( \frac{p}{p-1} \right)^{1/p} \cdot 2^{1+j/p} \cdot (\beta + 1). \)

Note that the construction of \( \widehat{t} \) in the above lemma guarantees that \( n^{-1/p} \leq \widehat{t} \leq 1 \)

Equipped with the above, we focus on the following problem for the rest of the section.

**Definition 2.5.5.3 (Adjusted mixed \((2, p)\)-norm problem).** Let \( p \geq 2 \), \( \mathbf{\Delta} \in \mathbb{R}^{n \times d} \), \( \mathbf{b} \in \mathbb{R}^d \)
and \( \mathbf{t} \in \mathbb{R}^n \) with \( n^{-1/p} \leq \mathbf{t} \leq 1 \) such that the optimal value of the following problem is at most one.

\[
\min \gamma_p(\mathbf{t}, \mathbf{\Delta}),
\]
\[
\text{s.t. } \mathbf{A}^\top \mathbf{\Delta} = \mathbf{b}.
\]

Then we call this problem an adjusted mixed \((2, p)\)-norm problem.

Note that in our case, \( \mathbf{\Delta} \) is the matrix \( \mathbf{A} \) concatenated with the gradient vector \( \mathbf{g} \), and \( \mathbf{b} \) is the vector zero concatenated by an adjusted version of the value \( \mathbf{g}^\top \mathbf{\Delta} \) for the optimal
solution. Then our goal is to find a solution with $\gamma_p(t, \Delta) < \beta$ for the above problem for some constant $\beta > 1$.

Similar to the mixed $(2, \infty)$-norm minimization, we solve a series of weighted linear regression problems of the form explained in Section 2.5.3. Our main contributions are two folds. We show that the algorithm of [8] outputs an approximate and almost feasible solution under fixed-point arithmetic with appropriate bit complexity. Moreover, we show that by using our inverse maintenance technique for the sparse solver, the running time improves beyond the current matrix multiplication time for poly-conditioned sparse matrices.

**Lemma 2.5.5.4 ([8]).** Let $p \geq 2$, $t, w, r \in \mathbb{R}^n$ with $t_j \geq n^{-1/p}$, $r_j = (n^{1/p} t_j)^{p-2} + w_j^{p-2}$, for all $j \in [n]$. Let $A \in \mathbb{R}^{n \times d}$, $b \in \mathbb{R}^d$, and $b \Delta = \arg\min \Delta$: $A^T \Delta = b$, $\|\Delta\|_r^2$.

Moreover, suppose $\min_{\Delta, A^T \Delta = b} \gamma_p(t, A) \leq 1$. Then

1. $\|\Delta\|_2^2 \leq \|\Delta\|_r^2 \leq n^{(p-2)/p} + \|w\|_p^{p-2}$

2. $A^T |\nabla \gamma(n^{1/p} t, w)| \leq p \gamma(n^{1/p} t, w)^{(p-1)/p} + p \cdot n^{(p-2)/2} \gamma(n^{1/p} t, w)^{1/2}$.

Note that if we replace $\hat{\Delta}$ with $\bar{\Delta}$ that is close to $\hat{\Delta}$ according to Lemma 2.1.2.3, then the bounds hold by multiplying an appropriate constant with the right-hand side.

A proof similar to [8] implies the following about the growth of the potential function $\gamma_p(n^{1/p} t, w^{(i,k)})$. Note that the main difference between this and the result of [8] is that our solution to the weighted linear regression problem has some error.

**Lemma 2.5.5.5 ([8]).** Let $p \geq 2$ and $i, k$ be nonnegative integers. Let $\varepsilon$ be the error of solving the weighted linear regression problems. Given $a^{p-1} \tau \leq n^{(p-1)/p}$ and $k \leq$
\[ n^{2/p} \beta^{-2/(p-2)} \rho^2, \]

\[ \gamma_p(n^{1/p} t, w^{(i,k)}) \leq (1 + \epsilon)^p \left( p^2 2^p \alpha + n^{1/p} \right)^p \exp \left( \frac{\zeta k}{n^{2/p} \beta^{-2/(p-2)} \rho^2} \right), \]

where \( \zeta := \frac{p^2 4^p}{p^{2p}} \left( (p^2 2^p + 1)^p + 1 \right) \) is just a function of \( p \).

A direct application of Lemma 2.5.5.4 and noting that \( w^{(i,k)} \leq \gamma_p(n^{1/p} t, w^{(i,k)})^{1/p} \) implies the following.

**Lemma 2.5.5.6 ([8]).** Let

\[ \Lambda^{(i,k)} = \arg \min_{\bar{\Lambda} \in \mathbb{B}} \| \Lambda \|^2_{r^{(i,k)}}. \]

Then \( \left\| \Lambda^{(0,0)} \right\|^2_{r^{(0,0)}} \geq \left\| \bar{\Lambda} \right\|_{r^{(i,k)}}^2 \), and

\[ \left\| \Lambda^{(i,k)} \right\|^2_{r^{(i,k)}} \leq n^{(p-2)/p} + \gamma_p(n^{1/p} t, w^{(i,k)})^{(p-2)/p}. \]

Since Lemmas 2.5.5.5 and 2.5.5.6 imply a bound on the growth of \( \left\| \Lambda^{(i,k)} \right\|^2_{r^{(i,k)}} \), if we show that in width reduction steps, it grows larger, then we have a bound on the maximum number of width reduction steps. The following shows that this function grows large in the width reduction step.

**Lemma 2.5.5.7 ([8]).** Consider a width reduction step in Algorithm 13, i.e., \( \bar{\Lambda}^{(i,k)} \|_{r^{(i,k)}} > \tau \).

Let \( q \geq 1 \gamma_p(n^{1/p} t, w^{(i,k)}) \leq q n, \tau^2/p \geq 2q \cdot n^{(p-2)/p} \beta^{-1}, \) and \( \tau \geq 10q \cdot \rho^{p-2} n^{(p-2)/p} \).

Moreover let

\[ \Lambda^{(i,k)} = \arg \min_{\bar{\Lambda}} \| \Lambda \|^2_{r^{(i,k)}}. \]

Then

\[ \left\| \Lambda^{(i,k+1)} \right\|^2_{r^{(i,k+1)}} \geq \left\| \Lambda^{(i,k)} \right\|^2_{r^{(i,k)}} \cdot \left( 1 + q \frac{\tau^2/p}{n^{(p-2)/p}} \right). \]

Moreover, for regular steps,

\[ \left\| \Lambda^{(i+1,k)} \right\|^2_{r^{(i+1,k)}} \geq \left\| \Lambda^{(i,k)} \right\|^2_{r^{(i,k)}}. \]
Note that in the above lemma \( q \) is a function of only \( p \) and comes from Lemma 2.5.5.5. Now directly combining Lemmas 2.5.5.5 to 2.5.5.7 gives the following bound for the number of iterations of Algorithm 13.

\[
O_p \left( n^{(p-2)/(3p-2)} \log^{p/(3p-2)} \left( n \left\| \| \mathbf{A} \|_2^2 / \| \mathbf{b} \|_2^2 \right\| \right) \right).
\]

The last piece is to bound the number and distribution of changes in the vector \( \mathbf{r} \). Then we can use our data structure results to give the desired running time bounds for both the sparse and dense cases. Note that even though the following result of [8] is with respect to exact solutions for the weighted linear regression problems, since we have the guarantee of \( \| \mathbf{A} - \tilde{\mathbf{A}} \|_2 \leq \epsilon \| \mathbf{A} \|_r \), for \( \tilde{\mathbf{A}} = \arg \min_{\mathbf{A} : \mathbf{A} \mathbf{b} = \mathbf{b}} \| \mathbf{A} \|_r \), from Lemma 2.1.2.3, we can guarantee that the error is small enough so that no constant factor change happens due to the error of the regression solution over the course of the algorithm.

**Theorem 2.5.5.8** ([8]). Let \( \ell_{e,\eta} \) be the number of indices \( j \) that are added to \( S \) at iteration \( e := i+k \) (where \( i \) and \( k \) are the numbers of regular and width-reduction steps, respectively) due to changes between \( 2^{-\eta} \) and \( 2^{-\eta+1} \) in Algorithm 13. Let

\[
T + K = \tilde{\Theta}_p \left( n^{p^2/(3p-2)} \log^{p/(3p-2)} \left( n \left\| \| \mathbf{A} \|_2^2 / \| \mathbf{b} \|_2^2 \right\| \right) \right)
\]

be the number of iterations (consisting of \( T \) regular steps and \( K \) width reduction steps). Then

\[
\sum_{e=1}^{T+K} \ell_{e,\eta} = \begin{cases} 
0 & \text{if } 2^\eta > T + K \\
\tilde{O}_p \left( n^{p^2/(3p-2)} \log^{p/(3p-2)} \left( n \left\| \| \mathbf{A} \|_2^2 / \| \mathbf{b} \|_2^2 \right\| \right) 2^{2\eta} \right) & \text{otherwise.} 
\end{cases} \tag{2.5.30}
\]

Now note that for \( q \) iterations, only \( \eta \) with \( 2^{\eta+1} < q \) can cause an index to be added to the set \( S \).

**Theorem 2.1.1.6** (continuing from p. 30). Let \( \mathbf{A} \in \mathbb{R}^{n \times d} \) be a matrix with condition number
bounded by $\kappa$, and $b \in \mathbb{R}^d$ be a vector with the bit complexity bounded by $\log(\kappa)$. Let $x^* = \arg \min_{Ax = b} \|x\|_p^p$. Let $m \leq n^{(\omega-2)/(\omega-1)}$ be the number of blocks in the block Krylov matrix used by the sparse linear system solver. For $2 \geq p$, there is an algorithm that finds $\widehat{x}$ such that $\|\pi_A(\widehat{x} - x^*)\|_2 \leq \epsilon \|\pi_Ax^*\|_2$ and

$$\|\widehat{x}\|_p^p \leq (1 + \epsilon) \|x^*\|_p^p$$

in time

$$\tilde{O}_p \left( \left( n^\omega + n^{7/3} \log(1/\epsilon) \right) \log^2(1/\epsilon) \log^{1.5}(\kappa/\epsilon) \right).$$

Moreover, for sparse matrices, there is an algorithm that returns an output with the same guarantees with probability at least $1 - n^{-10}$ in time

$$\tilde{O}_p \left( \left( n^\omega m^{7/3-\omega} + \text{NNZ}(A) \cdot m^{4/3} \cdot n \cdot \log(1/\epsilon) + n^{7/3} \cdot \log(1/\epsilon) \right) \log^{2.5}(\kappa/\epsilon) \log^2(1/\epsilon) \right).$$

Proof. First, by Lemma 2.5.1.4, the solution to the linear regression problem is polynomially close to the solution of the $p$-norm problem. Therefore by Lemma 2.5.1.6, we only need to solve $O_p(\log(n/\epsilon))$ instances of the residual problem to constant approximation. To do so by Lemmas 2.5.2.1 and 2.5.2.2, we only need to solve $O_p(\log(n/\epsilon) \log(n/\epsilon))$ instances of the smoothed $p$-norm minimization problems to constant factor approximation. Then Lemma 2.5.5.2 implies that to approximately solve each such instance, we only need to solve an adjusted smoothed $p$-norm minimization problem to constant factor approximation.

Now note that by Lemmas 2.5.5 to 2.5.5.7, in Algorithm 13, the number of width-reduction steps is bounded by $\tilde{O}_p(n^{2/p} \beta^{-2/(p-2)} \rho^2)$. Therefore by Lemma 2.5.5.5 and construction, Algorithms 12 and 13 output a vector $x$ such that

$$\gamma_p(n^{1/p}t, x) \leq \gamma_p(n^{1/p}t, w^{(T,K)}) = O_p(1) \cdot n.$$
where \( T \) and \( K \) are the numbers of regular steps and width-reduction steps, respectively. Therefore

\[
\gamma_p(t, n^{-1/p} x) = n^{-1/p} \gamma_p(n^{1/p} t, x) = O_p(1).
\]

Thus the output of the algorithm is a constant-factor approximation to the smoothed \( p \)-norm problem.

We now bound the time complexity of Algorithms 12 and 13 for both the dense and the sparse case. We first consider the dense case. By Lemmas 2.5.5.5 to 2.5.5.7, the number of iterations of the algorithm is

\[
\tilde{O}_p \left( n^{(p-2)/(3p-2)} \log^{p/(3p-2)} \left( n \left\| A \right\|_2^{2} / \left\| B \right\|_2^{2} \right) \right)
\]

which since \( p/(3p-2) \leq 0.5 \) for \( p \geq 2 \) is

\[
\tilde{O}_p \left( n^{(p-2)/(3p-2)} \log^{0.5} \left( \frac{\varepsilon}{\epsilon} \right) \right).
\]

In each iteration, we iteratively solve a constrained weighted regression problem by accessing a precondition. since \( \|w\|_p \leq \|w\|_p \leq \gamma_p(n^{1/p} t, w) \), by Lemma 2.5.5.5, \( \|r\|_{\infty} \leq \poly(n^p) \). Therefore by Lemma 2.1.2.3, each constrained weighted regression problem is solver in

\[
\tilde{O}_p \left( n^{7/3} \log(1/\epsilon) \log(1/\epsilon) \log(1/\epsilon) \right)
\]

for solving the constrained weighted regression problems given the preconditioner.

We now bound the running time of inverse maintenance. Consider the cost of inverse maintenance for updates that come from changes that are between \( 2^{-\eta} \) and \( 2^{-\eta+1} \). By Theorem 2.3.1.1, the cost of an update of rank \( r \) is \( O(MM(n, n, r) \log(\kappa/\epsilon)) \). Therefore by Theorem 2.5.5.8, and because we only need to consider \( \eta \)'s with \( (T + K)2^{-\eta} > 2 \) (larger \( \eta \)'s do not cause a constant-factor change over the course of the algorithm), the total cost of inverse maintenance over the course of the algorithm is

\[
\sum_{\eta=0}^{p-2} \frac{\log(n)}{3p-2} \sum_{T+K} \tilde{O}_p(MM(n, n, \ell_{\epsilon, \eta}) \log(\kappa/\epsilon)).
\]
Since 

$$MM(n, n, \ell_{e, \eta}) = \tilde{O}(n^2 \omega^{-2})$$,

and because we only perform inverse maintenance updates that come from the changes between $2^{-\eta} - 1$ and $2^{-\eta+1}$ once every $2^n$ iterations, by Theorem 2.5.5.8, the total cost of inverse maintenance is

$$\tilde{O}_p \left( n \omega^{-2} \log(n) \cdot \sum_{\eta=0}^{p-2} \tilde{O}_p \left( n^{(p-2)/(3p-2)} \right) \right),$$

Since there are only $O(\log(n))$ many different $\eta$, the total cost of inverse maintenance (above) is

$$\tilde{O}_p \left( n^\omega \log(n) \cdot \sum_{\eta=0}^{p-2} \tilde{O}_p \left( n^{(p-2)/(3p-2)} 2^{\eta(3(\omega-2)-1)} \right) \right).$$

Combining this with the cost of solving the constrained weighted regression problems and considering the number of residual problems and smoothed $p$-norm problems we solve gives the final running time for the dense case.

We now analyze the sparse case. First, note that for the sparse case, the randomness only comes from the probability of failure of the inverse operator in Theorem 2.3.2.1. Note that even if this reconstruction happens in every iteration, by the above discussion regarding
the number of iterations, the probability of failure is less than

\[ \tilde{O}_p(n^{-20}n^{(p-2)/(3p-2)} \log^2(1/\epsilon) \log^{0.5}(\kappa/\epsilon)). \]

Since \(1/\epsilon\) and \(\kappa\) are at most exponential in \(n\), this implies that the total failure probability is bounded by \(n^{-10}\) for large enough \(n\).

We trigger the reconstruction of the inverse operator once every \(\left\lfloor n^{(p-2)/(3p-2)}/m^{1/3} \right\rfloor\) iterations in Algorithm 13. Therefore by Theorem 2.3.2.2, the total cost for such reconstructions is

\[ \tilde{O}_p \left( \left(\text{NNZ}(A) \cdot m \cdot n + n^\omega m^{2-\omega} \right) \log^2(\kappa/\epsilon) \cdot m^{1/3} \log^{p/(3p-2)} \left( n \|A\|_2^2 / \|b\|_2^2 \right) \right), \]

which is

\[ \tilde{O}_p \left( \left(\text{NNZ}(A) \cdot m^{4/3} \cdot n + n^\omega m^{7/3-\omega} \right) \log^{2.5}(\kappa/\epsilon) \right). \]

The other way Algorithm 13 might trigger reconstruction of the inverse operator is that the sparse data structure (Theorem 2.3.2.2) receives an update of rank greater than \(n/m\). Note that since we force a reconstruction once every \(\left\lfloor n^{(p-2)/(3p-2)}/m^{1/3} \right\rfloor\) iterations, the only \(\eta\)'s that can trigger this second kind of construction should satisfy \(2^\eta < \left\lfloor n^{(p-2)/(3p-2)}/m^{1/3} \right\rfloor\).

The cost for such reconstructions is then

\[ \tilde{O}_p \left( \left(\text{NNZ}(A) \cdot m \cdot n + n^\omega m^{2-\omega} \right) \log^2(\kappa/\epsilon) \cdot \frac{n^{(p+2)/(3p-2)}2^{2\eta}}{n/m} \log^{p/(3p-2)} \left( n \|A\|_2^2 / \|b\|_2^2 \right) \right). \]

Since this is increasing in \(\eta\), and

\[ \frac{n^{(p+2)/(3p-2)}n^{2(p-2)/(3p-2)}m^{2/3}}{n/m} = m^{1/3}, \]
the total cost of this kind of reconstruction is also

\[ \tilde{O}_p \left( \left( \text{NNZ}(A) \cdot m^{4/3} \cdot n + n^\omega m^{7/3-\omega} \right) \log^{2.5}(\kappa/\epsilon) \right). \]

The final part of the inverse maintenance running time for the sparse case is when a Woodbury-type update happens, which only occurs when the rank of the update is less than \( n/m \). Note that in this case, by Theorem 2.3.2.2, and since \( (\cdot)^{\omega-2} \) is a concave function and \( p/(3p-2) < 0.5 \), for any \( \eta \), the cost is

\[ \tilde{O}_p \left( \left( \text{NNZ}(A) \cdot m^2 \cdot n^{(p+2)/(3p-2)} 2^{2\eta} + n^2 n^{(p-2)/(3p-2)} 2^{-\eta} \left( n^{4/(3p-2)3\eta} \right)^{\omega-2} \right) \cdot \log^{1/2}(\kappa/\epsilon) \log^2(\kappa/\epsilon) \right). \]

Since we only need to consider \( \eta \) such that \( 2^\eta < \left\lfloor n^{(p-2)/(3p-2)} / m^{1/3} \right\rfloor \), and this is increasing in \( \eta \), the total cost of these updates is

\[ \tilde{O}_p \left( \left( \text{NNZ}(A) \cdot m^{4/3} \cdot n + n^\omega m^{7/3-\omega} \right) \log^{2.5}(\kappa/\epsilon) \right). \]

Finally, by Theorem 2.3.2.2 and Lemma 2.1.2.3, the cost of solving constrained weighted regression problems is

\[ \tilde{O}_p \left( \left( \text{NNZ}(A) \cdot m^2 + n^2 \right) \log^2(\kappa/\epsilon) \cdot \log(1/\epsilon) \cdot n^{(p-2)/(3p-2)} \log^{p/(3p-2)} \left( n \| \mathbf{A} \|_2^2 / \| \mathbf{b} \|_2^2 \right) \right). \]

Now note that since \( (p-2)/(3p-2) < 1/3 \), and \( m < n \), this is

\[ \tilde{O}_p \left( \left( \text{NNZ}(A) \cdot m^{4/3} \cdot n + n^{7/3} \right) \log^2(\kappa/\epsilon) \cdot \log(1/\epsilon) \cdot \log^{1/2}(\kappa/\epsilon) \right). \]

Combining these running times with the number of residual problems and smoothed \( p \)-norm problems we have to solve gives the desired result. \( \square \)
Theorem 2.1.1.6 implies that for \( \text{NNZ}(A) = O(n) \), polynomially bounded \( \kappa/\epsilon \), and current value of \( \omega \), the running time is \( \tilde{O}_p(n^{2.363}) \).

2.6 Low-Rank Matrices

In this section, we first address the problem of \( p \)-norm minimization with low-rank matrices. We show that the matrix can be concatenated with a small multiple of the identity matrix, and this only slightly changes the solution.

**Lemma 2.1.4.3.** Let \( p \geq 2 \), \( A \in \mathbb{R}^{n \times d} \), \( b \in \mathbb{R}^d \), \( n \geq d \), such that the smallest nonzero singular value of \( A \) is equal to \( \sigma > 0 \). Moreover let \( 0 < \epsilon_1 < 1 \) and \( \epsilon_2 = \epsilon_1 \cdot \frac{\sigma}{2d^{(p-2)/2p}} \). Let

\[
\overline{A} = \begin{bmatrix} A \\ \epsilon_2 I \end{bmatrix}.
\]

Moreover let \( 0 < \epsilon_3 < 1 \), \( x^* \in \mathbb{R}^n \) and \( \overline{x} \in \mathbb{R}^{n+d} \) such that

\[
x^* = \arg \min_{x: A^\top x = b} \|x\|_p^p, \quad \|\overline{x}\|_p^p \leq (1 + \epsilon_3) \min_{x: A^\top x = b} \|x\|_p^p, \quad \text{and} \quad \|A^\top \overline{x} - b\|_2 \leq \epsilon_3.
\]

Let \( \overline{x} \in \mathbb{R}^d \) be a vector with entries equal to the first \( n \) entries of \( \overline{x} \). Then

\[
\|A^\top \overline{x} - b\|_2 \leq \epsilon_3 + \epsilon_1 \cdot \|b\|_2, \quad \text{and} \quad \|\overline{x}\|_p^p \leq (1 + \epsilon_3) \|x^*\|_p^p.
\]

**Proof.** First note that for any \( x \in \mathbb{R}^n \) such that \( A^\top x = b \), a padded with zero version \( \overline{x} \in \mathbb{R}^{n+d} \) of \( x \) satisfies \( \overline{A}^\top \overline{x} = b \). In addition \( \|x\|_p^p = \|\overline{x}\|_p^p \). Therefore

\[
\min_{x: A^\top x = b} \|x\|_p^p \leq \|x^*\|_p^p.
\]

Therefore

\[
\|\overline{x}\|_p^p \leq \|\overline{x}\|_p^p \leq (1 + \epsilon_3) \min_{x: A^\top x = b} \|x\|_p^p \leq (1 + \epsilon_3) \|x^*\|_p^p.
\]
Now let

\[ y^* = \arg \min_{y : A^T y = b} \|y\|_2^2. \]

Let \( z \in \mathbb{R}^d \) be a vector with entries equal to the last \( d \) entries of \( \tilde{x} \). We have

\[ \overline{A}^T \tilde{x} = A^T \tilde{x} + \varepsilon_2 z. \]

Therefore since \( \| \overline{A}^T \tilde{x} - b \|_2 \leq \varepsilon_3 \), by triangle inequality

\[ \| A^T \tilde{x} - b \|_2 = \| A^T \tilde{x} + \varepsilon_2 z - \varepsilon_2 z \|_2 \leq \varepsilon_3 + \| \varepsilon_2 z \|_2 \]

Moreover note that \( \| \tilde{x} \|_p = \| \tilde{x} \|_p^p + \| z \|_p^p \). Therefore \( \| z \|_p^p \leq (1 + \varepsilon_3) \| x^* \|_p^p \). By Holder’s inequality and definition of \( x^*, y^* \), we have

\[ \| z \|_2^p \leq d^{(p-2)/2} \| z \|_p^p \leq d^{(p-2)/2} \cdot (1 + \varepsilon_3) \cdot \| x^* \|_p^p \leq d^{(p-2)/2} \cdot (1 + \varepsilon_3) \cdot \| y^* \|_p^p \]

Now note that \( y^* = (A^T)^\dagger b \), since the \((A^T)^\dagger b\) is the solution to \( A^T y = b \) that has the minimum 2-norm [109]. Moreover since \( 0 < \varepsilon_3 < 1 \), and \( p \geq 2 \), \((1 + \varepsilon_3)^{1/p} < 2 \). Therefore

\[ \| z \|_2 \leq 2 \cdot d^{(p-2)/2p} \| y^* \|_2 \leq 2 \cdot d^{(p-2)/2p} \| (A^T)^\dagger \|_2 \| b \|_2 \leq 2 \cdot d^{(p-2)/2p} \| b \|_2^2. \]

Thus

\[ \| A^T \tilde{x} - b \|_2 \leq \varepsilon_3 + \| \varepsilon_2 z \|_2 \leq \varepsilon_3 + \varepsilon_1 \cdot \| b \|_2. \]

\[ \square \]
CHAPTER 3
SYMMETRIC FACTORIZATIONS OF HANKEL MATRICES

In this chapter, we present two conjectures regarding the running time of computing symmetric factorizations for a Hankel matrix $H$ and its inverse $H^{-1}$ as $BB^*$ under fixed-point arithmetic. If solved, these would result in a faster-than-matrix-multiplication algorithm for solving sparse poly-conditioned linear programming problems, a fundamental problem in optimization and theoretical computer science. To justify our proposed conjectures and running times, we show weaker results of computing decompositions of the form $BB^* - CC^*$ for Hankel matrices and their inverses with the same running time. In addition, to promote our conjectures further, we discuss the connections of Hankel matrices and their symmetric factorizations to sum-of-squares (SoS) decompositions of single-variable polynomials.

3.1 Introduction

Linear system solvers are a workhorse of the modern approach to optimization in which a linear system is solved in each iteration. This approach has been adapted for many problems ranging from graph problems [38], to $p$-norm regression [8], and linear programming [21, 22, 6]. If the linear systems in the problem have a special structure, then the structure can usually be exploited to obtain faster algorithms. This has probably been best exemplified by near-linear time Laplacian solvers that have led to improved running times in many graph problems [41, 42, 43, 44].

Solving a general linear system and various factorization of matrices can be done in $O(n^3)$ arithmetic (or field) operations. This can be improved using fast matrix multiplication techniques to $O(n^\omega)$, where $\omega < 2.373$ is the matrix multiplication exponent [16, 15, 14]. For solving linear systems with structured matrices such as Hankel and Toeplitz, fast algorithms with $O(n^2)$ arithmetic operations have been presented [110, 111]. This can
be improved to algorithms with $\tilde{O}(n)$ arithmetic operations. These are called super fast solvers [112, 113, 114, 115, 116]. These are based on finding a representation of the inverse that has $\tilde{O}(n)$ size (for example the inverse is constructed by shifting and adding a rank two matrix that can be presented by 4 vectors). The representation is then applied to the response vector of the linear system, for example, using fast Fourier transform (FFT) techniques [117]. Note that in such super fast algorithms, the inverse is never written explicitly since it costs $\Omega(n^2)$ to write an $n$-by-$n$ matrix explicitly.

Hankel matrices are a special class of structured matrices with many connections to other structured matrices such as Toeplitz, generalized Cauchy, and Vandermonde matrices [111, 114]. They also have many applications in theoretical computer science, including solving sparse linear systems [118, 107, 108, 45, 46, 119] (which itself has applications in improving runtime bounds for convex optimization algorithms [6, 7, 77, 8, 120]) and sum-of-squares (SoS) decomposition of single variable polynomials [121, 122, 123].

A recent breakthrough of Peng and Vempala [45] has shown that a poly-conditioned sparse linear system can be solved faster than matrix multiplication time by using block-Krylov methods. The high-level idea is to form a random block-Hankel matrix from the input matrix and then solve a linear system for this Hankel matrix instead. Although the bit complexity of this Hankel matrix is considerably more than the bit complexity of the input matrix (by a factor of $m < n^{0.25}$), Peng and Vempala showed, with a careful analysis, that the number of bit operations of their algorithm is $o(n^\omega)$ for any $\omega > 2$. Note that the algorithm of [45] does not generate an explicit inverse but instead generates a linear operator (an implicit inverse) that can be applied to a vector to solve the linear system.

Since the seminal works of Karmarkar [21] and Vaidya [22] on solving linear programs (LPs) using interior point methods (IPMs) maintaining the inverse of a matrix that goes under low-rank updates has been an important tool in improving the running time of algorithms for optimization problems. This inverse maintenance is done using Sherman-Morrison-Woodbury identity (Fact 2.1.2.1) which is equivalent to solving a batch of linear
systems, i.e., computing $A^{-1}B$ for a matrix $B$ instead of $A^{-1}b$, which is solving one linear system.

Although the sparse solver of Peng and Vempala is faster than matrix multiplication for solving one linear system, for a batch of linear systems of size $n$ (i.e., $B$ is an $n \times n$ matrix), it is slower than direct methods that compute an explicit inverse that can directly be multiplied by $B$ [105, 17]. Despite this caveat, the sparse solver has been utilized to improve the running time of $p$-norm regression problems for sparse poly-conditioned matrices beyond matrix multiplication time [120]. This improvement crucially depends on the fact that $p$-norm regression, for fixed $p$, can be solved by an algorithm with $\tilde{O}(n^{1/3})$ iterations [8]. A main idea of [120] is to recompute the linear operator associated with the inverse whenever the rank of the update in Sherman-Morrison-Woodbury identity is large and causes the running time to go above $n^\omega$. Since the number of iterations is $\tilde{O}(n^{1/3})$, this recomputation only happens a few times and a total running time of $o(n^\omega)$ is achieved for $p$-norm regression.

This approach, however, does not work for linear programming problems since the IPMs used for these problems require $\Omega(n^{1/2})$ iterations. We provide more details for this issue in Section 3.3. Inspired by this, we propose two conjectures regarding the running time of computing symmetric factorizations of the form $BB^*$ for Hankel matrices and their inverses, where $B^*$ denotes the conjugate transpose of the matrix $B$. Due to general displacement structures that we will discuss later, these conjectures have implications for the block-Hankel matrix arising in the block-Krylov approach of [45]. In particular, the following are implied by our conjectures.

1. The first implication of our conjectures is an algorithm for solving a batch of poly-conditioned linear systems faster than [45]. We have computed the running times of solving a batch of linear systems for a matrix with polynomial condition number and $O(n)$ nonzero entries using the online tool of Brand [124] that uses the running times developed in [104]. This is illustrated in Table 3.1. For example, for a batch of size
Table 3.1: Comparison of running times of [45] with an algorithm implied by our conjecture for solving a batch of linear systems for a poly-conditioned matrix with $O(n)$ nonzero entries.

<table>
<thead>
<tr>
<th>Size of batch</th>
<th>$n^{0.95}$</th>
<th>$n^{0.96}$</th>
<th>$n^{0.97}$</th>
<th>$n^{0.98}$</th>
<th>$n^{0.99}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Running time by [45]</td>
<td>$n^{2.354}$</td>
<td>$n^{2.357}$</td>
<td>$n^{2.361}$</td>
<td>$n^{2.365}$</td>
<td>$n^{2.369}$</td>
</tr>
<tr>
<td>Running time implied by our conjectures</td>
<td>$n^{2.341}$</td>
<td>$n^{2.341}$</td>
<td>$n^{2.349}$</td>
<td>$n^{2.357}$</td>
<td>$n^{2.365}$</td>
</tr>
</tbody>
</table>

$n^{0.96}$ (i.e., computing $A^{-1}B$ where $A \in \mathbb{R}^{n \times n}$ and $B \in \mathbb{R}^{n \times n^{0.96}}$), our approach would give an improvement of $n^{0.016}$ in the running time.

2. Perhaps the most important implication of our conjectures is an algorithm that solves a linear program with a sufficiently sparse matrix $A$ with polynomial condition number faster than matrix multiplication time. The sufficient sparsity is $o(n^{\omega-1})$ nonzero entries. We discuss this in detail in Section 3.3.

3. In addition, the algorithm developed based on our conjectures improves the running time of the sparse $p$-norm regression algorithm developed in [120].

Outline. Motivated by these applications, we present necessary definitions and preliminaries for understanding our conjectures and results in Section 3.1.1. We then present our conjectures and corresponding results that justify them in Section 3.2. We discuss the applications of Hankel matrices and the implications of our conjectures, including the implications for solving sparse linear programs, in Section 3.3. We then provide a result regarding symmetric factorizations of Toeplitz matrices (which is used as a subprocedure for symmetric factorization of Hankel matrices) in Section 3.4. We present a key identity for Hankel matrices in Section 3.5 that allows us to design a recursive algorithm for symmetric factorization of them. We then present our results regarding the symmetric factorization of Hankel matrices and their inverses in Sections 3.6 and 3.7, respectively. We finally conclude in Section 3.8.
3.1.1 Notation and Preliminaries

We consider the entries of our matrices to be in a field \( \mathbb{F} \). This can be considered the field of reals \( \mathbb{R} \) or complex numbers \( \mathbb{C} \). For both of these, our factor matrices \( A \) and \( B \) are in \( \mathbb{C} \). Our results also extend to finite fields \( \mathbb{F} \). In this case, the entries of \( A \) and \( B \) are from an extension of field \( \mathbb{F} \) that contains the square root of all of the elements of \( \mathbb{F} \). For \( \mathbb{R} \) and \( \mathbb{C} \), we consider fixed-point arithmetic for computation and representing our numbers. In this case, we cannot necessarily represent the square root of our numbers with finitely many bits, but for a number \( a \) with \( \ell \) bits, we can find a number \( b \) with \( O(\ell) \) bits such that \( |b - \sqrt{a}| < 2^{-\ell} \). Therefore for matrices over \( \mathbb{R} \) and \( \mathbb{C} \), our symmetric factorizations have some small error.

For matrices over \( \mathbb{R} \) and \( \mathbb{C} \), we denote the Frobenius norm and the operator norm by \( \| \cdot \|_F \) and \( \| \cdot \|_2 \), respectively. Then we define the condition number of an invertible matrix \( A \) over \( \mathbb{R} \) or \( \mathbb{C} \) as \( \| A \|_2 \cdot \| A^{-1} \|_2 \), and we denote it by \( \kappa(A) \).

We denote the entry \((j, k)\) of a matrix \( M \) either by \( M_{j,k} \) or \( M(j, k) \). For natural numbers \( j_2 > j_1 \) and \( k_2 > k_1 \), we show the block of \( M \) with rows \( j_1, j_1 + 1, \ldots, j_2 \) and columns \( k_1, k_1 + 1, \ldots, k_2 \) with \( M_{j_1:j_2,k_1:k_2} \). The matrix consisting of rows \( j_1, \ldots, j_2 \) and all columns is denoted by \( M_{j_1:j_2,:} \). We denote the \( n \)-by-\( n \) identity matrix with \( I_n \) and if the dimension is clear from the context, we drop the subscript. We denote an \( m \)-by-\( n \) matrix of all zeros with \( 0_{m \times n} \) and if the dimensions are clear from the context, we drop the subscript. We denote the positive definite (Loewner) ordering by \( \preceq \). We denote the running time of multiplying an \( n \times m \) matrix with an \( m \times k \) matrix with \( MM(n, m, k) \). Then \( n^{\omega} = MM(n, n, n) \).

We denote the transposition of a matrix \( M \) by \( M^\top \) and its conjugate transposition by \( M^* \). Note that for real matrices, transposition and conjugate transposition are the same. We also denote the complex conjugate of a number \( a \in \mathbb{C} \) by \( a^* \). Moreover we define \( i = \sqrt{-1} \). For a matrix \( M \), we denote its real part and imaginary part by \( \text{real}(M) \) and \( \text{imag}(M) \), respectively. Note that both \( \text{real}(M) \) and \( \text{imag}(M) \) are real matrices and \( M = \text{real}(M) + i \cdot \text{imag}(M) \).

We use \( \tilde{O} \) notation to omit polylogarithmic factors in \( n \) and \( \ell \) from the complexity, i.e., for function \( f \), \( \tilde{O}(f) := O(f \cdot \log^c(n\ell)) \) where \( c \) is a constant. We denote the set \( \{1, \ldots, n\} \)
by \([n]\).

We extensively use the shift matrix \(\Delta_n \in \mathbb{F}^{n \times n}\) that is zero everywhere except on the entries under the diagonal for which it is one. For example,

\[
\Delta_4 = \begin{bmatrix}
0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0
\end{bmatrix}.
\]

When the dimension of \(\Delta_n\) is clear from the context, we omit the subscript and show the shift matrix by \(\Delta\). Multiplying a matrix from left by \(\Delta\) \((\Delta^T)\) shifts the rows of the matrix down (up) by one row and multiplying a matrix from right by \(\Delta\) \((\Delta^T)\) shifts the columns of the matrix left (right) by one column. A matrix \(M\) is symmetric if \(M = M^T\) and is Hermitian if \(M = M^*\). Also \(M\) is skew-symmetric if \(M^T = -M\). Let \(\mathbb{F}\) be a field and \(h = (h_1, \ldots, h_{2n-1}) \in \mathbb{F}^{2n-1}\) be a vector. Then the corresponding Hankel matrix \(H\) is defined as \(H_{ij} = h_{i+j-1}\). For example for \(n = 4\),

\[
H = \begin{bmatrix}
h_1 & h_2 & h_3 & h_4 \\
h_2 & h_3 & h_4 & h_5 \\
h_3 & h_4 & h_5 & h_6 \\
h_4 & h_5 & h_6 & h_7
\end{bmatrix}.
\]

For a vector \(T = (t_1, \ldots, t_n) \in \mathbb{C}^n\), where \(t_1 \in \mathbb{R}\), the corresponding Hermitian Toeplitz matrix \(T\) is defined as \(T_{i,j} = t_{j-i+1}\) if \(j \geq i\), and \(T_{i,j} = t_{i-j+1}^*\), otherwise. For example, the
Hermitian Toeplitz matrix corresponding to $(t_1, t_2, t_3, t_4)$ is

\[
T = \begin{bmatrix}
    t_1 & t_2 & t_3 & t_4 \\
    t_2^* & t_1 & t_2 & t_3 \\
    t_3^* & t_2^* & t_1 & t_2 \\
    t_4^* & t_3^* & t_2^* & t_1
\end{bmatrix}.
\]

Note that this can be considered for a general field by extending it using the polynomial root $x^2 + 1 = 0$. It is easy to check that for a Toeplitz matrix $T$, $T - \Delta T \Delta^T$ is of rank two, and for a Hankel matrix $H$, $\Delta H - H \Delta^T$ has rank two. These are called the displacement rank of Toeplitz and Hankel matrices. The general definitions are as the following.

**Definition 3.1.1.1 (Displacement rank).** Let $M, U, V \in \mathbb{F}^{n \times n}$. The **Sylvester-type displacement rank** of $M$ with respect to $(U, V)$ is equal to the rank of $UM - MV$. The **Stein-type displacement rank** of $M$ with respect to $(U, V)$ is equal to the rank of $M - UMV$.

For example a Hankel matrix $H$ has a Sylvester-type displacement rank of two with respect to $(\Delta, \Delta^T)$. This allows us to define the displacement rank for block-Hankel matrices of the following form as well.

\[
H = \begin{bmatrix}
    H_1 & H_2 & H_3 & H_4 \\
    H_2 & H_3 & H_4 & H_5 \\
    H_3 & H_4 & H_5 & H_6 \\
    H_4 & H_5 & H_6 & H_7
\end{bmatrix},
\]

where each $H_i$ is an $s \times s$ matrix. Then $H$ has a Sylvester-type displacement rank of $2s$ with
respect to \((U, U^T)\), where

\[
U = \begin{bmatrix}
0_{s \times s} & 0_{s \times s} & 0_{s \times s} & 0_{s \times s} \\
I_s & 0_{s \times s} & 0_{s \times s} & 0_{s \times s} \\
0_{s \times s} & I_s & 0_{s \times s} & 0_{s \times s} \\
0_{s \times s} & 0_{s \times s} & I_s & 0_{s \times s}
\end{bmatrix}.
\]

Moreover, the inverse of a Hankel matrix is not Hankel but it has a Sylvester-type displacement rank of two with respect to \((\Delta, \Delta^T)\). Similarly the inverse of a Toeplitz matrix is not Toeplitz but it has a Stein-type displacement rank of two with respect to \((\Delta, \Delta^T)\).

Note that multiplying a vector by a Hankel or Toeplitz matrix can be done in \(\tilde{O}(n)\) time using FFT techniques [26, Chapter 30] due to their connections to single-variable polynomials. Finally, the following illustrates the connection between inverse maintenance and solving a batch of linear systems.

### 3.2 Results and Conjectures

Our first conjecture is about computing a symmetric factorization of positive definite Hankel matrix \(H\) as \(BB^*\) in linear time. Since \(B\) is at least \(n \times n\) (for a full-rank Hankel matrix), we do not require outputting \(B\) explicitly. Instead, the output should be an implicit representation of size \(\tilde{O}(n \cdot \ell)\) that describes \(B\). Note that this is similar to the way that Hankel matrices are described as well. For example, if we give \((h_1, \ldots, h_7)\) in (Equation 3.1.1), then the corresponding Hankel matrix is completely described and this representation has a linear size in \(n\).

**Conjecture 3.2.0.1.** Let \(H \in \mathbb{R}^{n \times n}\) be a positive definite Hankel matrix with bit complexity \(\ell\). There exists an algorithm that finds a representation of a matrix \(B\) with \(n\) rows, \(\tilde{O}(n)\) columns, and bit complexity \(\ell\) in time \(\tilde{O}(n \cdot \ell)\) such that \(\|H - BB^*\|_F < \frac{1}{2^\ell}\).

Our conjecture over finite fields would require \(B\) such that \(H = BB^*\). In this case, we
assume the field operations are performed in $O(1)$ time, and therefore we require a running time of $\tilde{O}(n)$. For matrices over $\mathbb{C}$, we also require $\|H - BB^*\|_F < \frac{1}{2^\ell}$. To justify our conjecture, we provide an algorithm that runs in the specified running time and computes a representation of a factorization of the form $BB^* - CC^*$.

**Theorem 3.2.0.2.** Let $H \in \mathbb{R}^{n \times n}$ be a Hankel matrix with bit complexity $\ell$. There exists an algorithm that finds a representation of matrices $B$ and $C$, each with $n$ rows, $O(n \log n)$ columns, and bit complexity $\ell$ in time $\tilde{O}(n \cdot \ell)$ such that $\|H - (BB^* - CC^*)\|_F < \frac{1}{2^\ell}$.

Since Hankel matrices are symmetric, Theorem 3.2.0.2 does not require the positive definite condition. Our algorithm gives similar bounds and running times for matrices over $\mathbb{C}$ and over finite fields it finds representations of $B$ and $C$ such that $H = BB^* - CC^*$.

The factorization of the form $BB^* - CC^*$ has been considered before for Toeplitz matrices and their inverses with the goal of solving linear systems with a Toeplitz matrix in linear time [125, 117, 112]. The positive semi-definiteness of $BB^*$ and $CC^*$ provides some stability properties for solving linear systems with a Toeplitz matrix [126]. These algorithms are related to the study of orthogonal polynomials and generally either use the Schur algorithm or Levinson algorithm to compute $B$ and $C$. We provide similar results for Toeplitz matrices with a simpler and more straightforward algorithm.

**Theorem 3.2.0.3.** Let $T \in \mathbb{R}^{n \times n}$ be a Hermitian Toeplitz matrix with bit complexity $\ell$. There exists an algorithm that finds a representation of matrices $B$ and $C$, each with $n$ rows, $O(n \log n)$ columns, and bit complexity $\ell$ in time $\tilde{O}(n \cdot \ell)$ such that $\|T - (BB^* - CC^*)\|_F < \frac{1}{2^\ell}$.

Theorem 3.2.0.3 is used as a subprocedure for Theorem 3.2.0.2. The simplicity of our algorithm for Toeplitz matrices allows us to use it in combination with a recursive algorithm that recursively decomposes a Hankel matrix to the sum of $\log(n)$ Toeplitz-like matrices to achieve our main result for decomposition of Hankel matrices.
Our second conjecture is about computing a symmetric factorization for the inverse of a positive definite Hankel matrix. Note that we do not require the inverse as input since the approach of [45] can obtain a representation of it that has a linear size in linear time.

**Conjecture 3.2.0.4.** Let \( H \in \mathbb{R}^{n \times n} \) be a positive definite Hankel matrix with bit complexity \( \ell \) and condition number bounded by \( 2^\ell \). There exists an algorithm that finds a representation of a matrix \( B \) with \( n \) rows, \( O(n) \) columns, and bit complexity \( \ell \) in time \( O(n^{\omega/2} \cdot \ell) \) such that \( \|H^{-1} - BB^*\|_F < \frac{1}{2^\ell} \).

Again for finite fields, we require \( H^{-1} = BB^* \) and a running time of \( O(n^{\omega/2}) \). Note the difference between the running time of Conjecture 3.2.0.1 and Conjecture 3.2.0.4. This is because of the running time that we can achieve for the factorization of the form \( BB^* - CC^* \) in the following result.

**Theorem 3.2.0.5.** Let \( H \in \mathbb{R}^{n \times n} \) be a Hankel matrix with bit complexity \( \ell \) and condition number bounded by \( 2^\ell \). There exists an algorithm that finds a representation of matrices \( B \) and \( C \), each with \( n \) rows, \( O(n \log n) \) columns, and bit complexity \( \ell \) in time \( O(n^{\omega/2} \cdot \ell) \) such that \( \|H^{-1} - (BB^* - CC^*)\|_F < \frac{1}{2^\ell} \).

The result of Theorem 3.2.0.5 is actually more general than the inverse of Hankel matrices. The algorithm we present can find such a factorization in the specified time for any matrix that has a Sylvester-type displacement rank of two with respect to \((\Delta, \Delta^T)\) and it can be generalized to block matrices as described in Section 3.1.1.

The main reason for the running time difference between Theorem 3.2.0.2 and Theorem 3.2.0.5 is the recursion in our algorithm. For Theorem 3.2.0.2, our recursion starts with the \( n \times n \) Hankel matrix and modifies it to a matrix with four blocks of size \( \frac{n}{2} \times \frac{n}{2} \) where each block itself is Hankel, i.e., the displacement rank of the blocks is the same as the larger matrix. It then continues this process for \( O(\log n) \) iterations. However, for general matrices with small Sylvester-type displacement rank, when we apply the recursion, the Sylvester-type displacement rank of the blocks is doubled. This forces us to stop the
recursion when the size of the blocks is $\sqrt{n}$ and results in the running time proportional to $n^{\omega/2}$.

3.3 Motivation and Related Work

Hankel matrices have many connections to Toeplitz matrices. One can see that reversing the order of rows or columns of a Hankel matrix results in a Toeplitz matrix and vice versa. Therefore solving a linear system for Toeplitz matrices implies a solver for Hankel matrices as well. Therefore many works have focused on Toeplitz matrices. However, there are some applications that are specifically directed to Hankel matrices. Examples are linear system solvers based on block Krylov matrices (that are used to solve linear systems with general poly-conditioned sparse matrices [45, 46]) and sum-of-squares (SoS) decomposition of single-variable polynomials.

Here we first discuss sparse linear system solvers based on block-Krylov methods in Section 3.3.1 and explain how our conjecture leads to faster algorithms for solving a batch of linear systems. Then in Section 3.3.2, we explain how this leads to a faster algorithm for solving sparse poly-conditioned linear programs faster than matrix multiplication time. We finally discuss the connection of Hankel matrices to the sum-of-squares (SoS) decomposition of single variable polynomials in Section 3.3.3.

3.3.1 Faster Sparse Linear System Solvers for Batch Problems

We start by describing the block-Krylov approach that has resulted in faster sparse linear system solvers for matrices over rational numbers [107, 108], fixed-point arithmetic [45, 46], and finite fields [119].
Linear system solvers based on block-Krylov matrices. To solve a linear system $Ax = b$, this approach forms a block Krylov matrix

$$K = \begin{bmatrix} G & AG & A^2G & \cdots & A^{m-1}G \end{bmatrix} \in \mathbb{R}^{n \times n},$$

where $G$ is a sparse $n$-by-$s$ random matrix, and $m \cdot s = n$. If the matrix $A$ is sparse, for example, its number of nonzero entries is $O(n)$, then $K$ can be formed quickly. Note that $A^{i+1}G$ can be obtained from $A^iG$ by multiplying it with $A$. More specifically, for $A$ with constant bit complexity, $K$ can be formed in time $\tilde{O}(\text{NNZ}_A \cdot s \cdot m^2) = \tilde{O}(\text{NNZ}_A \cdot n \cdot m)$, where the $\text{NNZ}_A \cdot s$ factor comes from the time that takes to multiply $A$ by an $n$-by-$s$ matrix. One of the factors of $m$ comes from the number of such matrix multiplications we need to perform and the other one comes from the bit-complexity of the resulting matrices, e.g., the entries of $A^{m-1}G$ need $\tilde{O}(m)$ bits. For a small enough $m$ (for example, $m \approx n^{0.01}$), $\tilde{O}(\text{NNZ}_A \cdot n \cdot m)$ is smaller than the matrix multiplication time if $\text{NNZ}_A \ll n^{\omega-1}$.

Then the inverse of $A$ is presented by $K(K^TAK)^{-1}K^T$. Note that for symmetric $A$, $K^TAK$ is a block-Hankel matrix of the following form

$$K^TAK = \begin{bmatrix} G^TAG & G^TA^2G & G^TA^3G & \cdots & G^TA^mG \\ G^TA^2G & G^TA^3G & G^TA^4G & \cdots & G^TA^{m+1}G \\ G^TA^3G & G^TA^4G & G^TA^5G & \cdots & G^TA^{m+2}G \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ G^TA^mG & G^TA^{m+1}G & G^TA^{m+2}G & \cdots & G^TA^{2m-1}G \end{bmatrix} \in \mathbb{R}^{n \times n}.$$ 

Note that the symmetry assumption for $A$ is not a limitation since we can instead consider the linear system $A^T Ax = A^Tb$, which has a symmetric matrix. One can think of this matrix as an $m$-by-$m$ Hankel matrix where each entry is a $s$-by-$s$ matrix with bit-complexity of $\tilde{O}(m)$. Therefore multiplying any two entries of this matrix together costs $\tilde{O}(s^\omega \cdot m)$. Moreover $K^TAK$ can be multiplied with an $n \times s$ matrix in time $\tilde{O}(s^\omega \cdot m^2)$ by using fast
Fourier transform (see [45] for details). Finally, note that $K^T A K$ can be formed in time $\tilde{O}(\text{NNZ}_A \cdot n \cdot m)$ similar to the approach we described above for computing $K$.

**Fast and super fast solvers for block-Hankel matrices.** To discuss the running time of inverting the block-Hankel matrix $K^T A K$ or applying the inverse to a block-matrix (or a vector), we need to consider the number of block operations. One can think of each block operation as multiplying two blocks of $K^T A K$ together. These blocks are $s \times s$ and have bit complexity $m$. So multiplying them by fast matrix multiplication [16] and using FFT to multiply the corresponding numbers in linear time results in a running time of $\tilde{O}(s^\omega \cdot m)$. Therefore an algorithm that takes $k$ block operations runs in time $\tilde{O}(s^\omega \cdot m \cdot k)$ with the assumption that the bit complexity stays the same during the algorithm.

Therefore fast solvers that need $m^2$ operations are slow for inverting $K^T A K$, since they result in a total cost of $s^\omega \cdot m^3 > n^\omega$. Thus one needs to use super fast solvers for the matrix $K^T A K$. Most of the classical super fast solvers are either based on orthogonal polynomials [117, 112] or based on the conversion of Hankel matrix to generalized Cauchy and hierarchically semi-separable (HSS) matrices [115, 116] that admit low-rank properties for off-diagonal blocks. The caveat of these methods is that they blow up the bit complexity of $L$ to at least $L^2$. This means an extra factor of $m$ in addition to $s^\omega \cdot m^2$ operations which again results in a total running time of more than $n^\omega$.

There has been another class of super fast solvers based on hierarchical Cholesky decomposition and Schur complements that classically were analyzed in the exact computation setting (for example, for matrices on finite fields) [118, 127]. Very recently, [45] analyzed such algorithms for real matrices in the fixed-point arithmetic and showed that such super-fast solvers only need to increase the bit complexity by polylogarithmic factors in $n$. This resulted in an algorithm with a total running time of $\tilde{O}(s^\omega m^2)$ for finding a representation of the inverse of $K^T A K$. This algorithm was one of the main building blocks that allowed [45] to go below matrix multiplication time. The representation of the inverse
of $K^T A K$ obtained from this approach is the product of two matrices $X$ and $Y^T$, i.e.,

$$(K^T A K)^{-1} \approx X Y^T. \quad (3.3.1)$$

$X$ and $Y$ are block matrices with a small displacement rank of $2s$. Therefore they can be applied to another matrix of size $n \times s$ with $\tilde{O}(m)$ block operations by utilizing FFT.

The caveat of this approach is that the bit complexity of matrices $X$ and $Y$ is $\Omega(m)$. Therefore although they can solve one linear system faster than matrix multiplication time, for any selection of parameters $s$ and $m$, there is a $0 < c < 1$ such that solving $n^c$ linear system with a common matrix $A$ takes more than the matrix multiplication time. This is strange since inverting the matrix $n$ using fast matrix multiplication takes $\tilde{O}(n^\omega)$ time and then the inverse can be applied to $n$ vectors in $\tilde{O}(n^\omega)$ time [105] and this does not need any sparsity properties. We now bound the running time of solving a batch of linear systems of size $r$ with [45] solver. To do so, we need the following lemma for the running time of applying the matrix $K$ to a matrix of size $n \times r$.

**Lemma 3.3.1.1.** Let $A \in \mathbb{R}^{n \times n}$ and $B \in \mathbb{R}^{n \times r}$. Let $G \in \mathbb{R}^{n \times s}$ be a matrix with $\tilde{O}(n)$ nonzero entries, where $1 \leq s \leq n$ is a divisor of $n$. Let $m = n/s$ and

$$K = \begin{bmatrix} G & AG & A^2G & \cdots & A^{m-1}G \end{bmatrix} \in \mathbb{R}^{n \times nr}.$$

Let bit complexity of $A$ and $G$ be $\ell$ and the bit complexity of $B$ be $m \cdot \ell$. Then $KB$ and $K^TB$ can be computed in $\tilde{O}(\text{NNZA} \cdot r \cdot m^2 \cdot \ell)$ time.

**Proof.** We first discuss he running time of computing $KB$. We partition the rows of $B$ to $m$
blocks $B_1, \ldots, B_m \in \mathbb{R}^{s \times n}$ as the following

$$
B = \begin{bmatrix}
B_1 \\
B_2 \\
\vdots \\
B_m
\end{bmatrix}.
$$

Then we have

$$
KB = \sum_{i=1}^{m} A^{i-1} GB_i.
$$

For $j \in [m]$, define

$$
M_j = \sum_{i=j}^{m} A^{i-j} GB_i.
$$

Therefore $KB = M_1$ and for $j \geq 2$,

$$
M_{j-1} = \sum_{i=j-1}^{m} A^{i-j+1} GB_i \tag{3.3.2}
$$

$$
= GB_{j-1} + \sum_{i=j}^{m} A^{i-j+1} GB_i
$$

$$
= GB_{j-1} + A \sum_{i=j}^{m} A^{i-j} GB_i
$$

$$
= GB_{j-1} + AM_j.
$$

Therefore given $M_j$, we can compute $M_{j-1}$ in time $\tilde{O}(mnr \cdot \ell + \text{NNZA} \cdot r \cdot (\ell + \ell_j))$, where $\ell_j$ is the bit complexity of $M_j$. The first term in the running time is for computing $GB_{j-1}$ and essentially follows from the fact that each column of $B_{j-1}$ can be multiplied by $G$ in $\tilde{O}(mn \cdot \ell)$ time because the bit complexity of $B$ is at most $m\ell$ and $G$ has $\tilde{O}(n)$ nonzero entries. The second term in the running time is for computing $AM_j$ and follows from a similar argument. Moreover $\ell_{j-1} = \tilde{O}(\max\{m \cdot \ell, \ell + \ell_j\})$.

Since $M_m = GB_m$, it can be computed in $\tilde{O}(mnr \cdot \ell)$ time and it has a bit complexity of
\( \tilde{O}(m \cdot \ell) \). Therefore by the recurrence relation (Equation 3.3.2), \( KB = M_1 \) can be computed in time

\[
\tilde{O}(\text{nnz} A \cdot r \cdot m^2 \cdot \ell)
\]

because the recursion only goes for \( m \) steps and the bit complexity of the intermediate matrices stays \( \tilde{O}(m \cdot \ell) \).

Computing \( K^T B \) is less complicated. Note that

\[
K^T B = \begin{bmatrix}
G^T B \\
G^T A^T B \\
G^T (A^T)^2 B \\
\vdots \\
G^T (A^T)^{m-1} B
\end{bmatrix}
\]

Given \( (A^T)^j B \), \( G^T (A^T)^j B \) and \( (A^T)^{j+1} B \) can be computed in time \( \tilde{O}(n \cdot r \cdot (\ell + \ell_j)) \) and \( \tilde{O}(\text{nnz} A \cdot r \cdot (\ell + \ell_j)) \), where \( \ell_j \) is the bit complexity of \( (A^T)^j B \). Therefore this gives a total running time of

\[
\tilde{O}(\text{nnz} A \cdot r \cdot m^2 \cdot \ell).
\]

\[\square\]

Assuming the bit complexity of the input matrix is constant and its condition number is \( \text{poly} n \), for a fixed \( m \) and \( r \), the total running time of applying the inverse operator of [45] to an \( n \times r \) matrix is the following.

\[
\tilde{O}(\text{nnz} A \cdot n \cdot m + n^\omega m^{2-\omega} + m^2 \cdot \text{MM}(\frac{n}{m}, \frac{n}{m}, r) + \text{nnz} A \cdot r \cdot m^2).
\]

(3.3.3)

The first term of (Equation 3.3.3) is for forming \( K \) and \( K^T A K \). The second term is finding the representation of the inverse. The third term is the running time of applying the inverse
of $K^TAK$ to an $n \times r$ matrix. The last term is for applying $K^T$ or $K$ to an $n \times r$ matrix (see Lemma 3.3.1.1). Note that for solving one linear system, (Equation 3.3.3) boils down to

$$\tilde{O}(\text{NNZA} \cdot n \cdot m + n^{\omega}m^{2-\omega}).$$

Then one can see that by taking $m = n \cdot (\text{NNZA})^{-1/(\omega-1)}$, a running time of

$$\tilde{O}(n^2(\text{NNZA})^{(\omega-2)/(\omega-1)})$$

is achieved, which is faster than matrix multiplication for all values of $\omega > 2$ and $\text{NNZA} < n^{\omega-1}$.

The running time of (Equation 3.3.3) is obtained by applying $K$, $(K^TAK)^{-1}$ and $K^T$ separately. Another approach is to take $X$ and $Y$ from (Equation 3.3.1) and compute $\bar{X} = KX$ and $\bar{Y} = KY$ using Lemma 3.3.1.1. Then the inverse of $A$ is given by $\bar{X}\bar{Y}^T$, where the bit complexity of $\bar{X}$ and $\bar{Y}$ is $O(m)$. Then solving a batch of linear systems of size $r$ by multiplying $\bar{X}$ and $\bar{Y}$ takes the following running time.

$$\tilde{O}(\text{NNZA} \cdot n \cdot m^2 + n^{\omega}m^{2-\omega} + m \cdot \text{MM}(n,n,r)).$$

(3.3.4)

The first term of (Equation 3.3.4) is from computing $\bar{X}, \bar{Y}$, which also dominates the running time of forming $K$ and $K^TAK$. The second term is for finding the representation of the inverse of $K^TAK$, and the last term comes from the running time of multiplying $\bar{X}$ and $\bar{Y}$ with an $n \times r$ matrix. Given a fixed $r$, one can optimize over the best value of $m$ for each of (Equation 3.3.3) and (Equation 3.3.4) and report the smaller running time. This is what we used for Table 3.1.

**Symmetric factorization of inverse operator for faster batch solves.** The main caveat of the approach of [45] is that the representation of the inverse has a bit complexity of $\Omega(m)$,
whether we use $\widetilde{XY}^{-1}$ representation or $KXY^TY$. This is the main reason that when applied to large batches, the running time of [45] becomes slower than direct methods. Here we present an approach based on our conjectures to obtain a representation of the inverse with small bit complexity.

Our approach is to write the inverse of $K^TAK$ as a symmetric factorization $XX^*$. In this case, the inverse of $A$ is represented as $(KX)(KX)^*$. Therefore we have

$$\|KX\|_F = \sqrt{\text{trace}((KX)(KX)^*)} = \sqrt{\text{trace}(A^{-1})} = \sqrt{\sum_{i=1}^{n} \lambda_i}, \quad (3.3.5)$$

where $\lambda_i$’s are the eigenvalues of $A^{-1}$. Therefore in the case where $\lambda_i$’s are poly($n$) (which is the assumption in [45, 46]), the absolute value of entries in the matrix $KX$ is bounded by poly($n$). Moreover, one can compute $KX$ using Lemma 3.3.1.1. Therefore, in this case, we can represent the inverse of $A$ as $\widetilde{XX}^*$, where $\widetilde{X} = KX$, and the bit complexity of entries of $\widetilde{X}$ is $\tilde{O}(1)$. Then the running time of solving a batch of linear systems of size $r$ becomes

$$\tilde{O}(\text{NNZ}A \cdot n \cdot m^2 + n^\omega m^{1-\omega/2} + \text{MM}(n,n,r)) \quad (3.3.6)$$

since $\widetilde{X}$ can be applied to an $n \times r$ matrix in time $\text{MM}(n,n,r)$. Note that we require the error bound of less than $1/2^\ell$ (which here be less than $1/2^m$) in Conjecture 3.2.0.1 and Conjecture 3.2.0.4 because the bit complexity of $K$ is $\tilde{O}(m)$ and this way we can guarantee that $(KX)(KX)^*$ is close to $A^{-1}$. Note that Conjecture 3.2.0.4 gives an algorithm that runs with $\tilde{O}(m^{\omega/2})$ block operations and uses numbers with the bit complexity of the input problem. Therefore Conjecture 3.2.0.4, if true, computes a representation of the matrix $X$ in time $\tilde{O}(s^\omega \cdot m^{1+\omega/2}) = \tilde{O}(n^\omega m^{1-\omega/2})$ such that

$$\|XX^* - K^TAK\|_F \leq \frac{1}{2^m}.$$ 

Since the bit complexity of this representation is $\tilde{O}(m)$, we can write down $X$ in time
\( \tilde{O}(n^2 \cdot m) \) and then use Lemma 3.3.1.1 to compute \( \tilde{X} = KX \) in time \( \tilde{O}(\text{NNZ}_A \cdot n \cdot m^2) \). This gives us the running time stated in (Equation 3.3.6), which is also the formula we used for our running time in Table 3.1. We next discuss how our approach results in a faster-than-matrix-multiplication time for solving linear programs with sparse and poly-conditioned matrices.

### 3.3.2 Solving Linear Programs Faster than Matrix Multiplication

Here we first give a simple explanation of the linear systems that are solved in each iteration of interior point methods (IPMs) for solving LPs. IPMs are the state-of-the-art approach for solving LPs. The seminal works of Karmarkar [21] and Vaidya [22] started the study of IPMs, and recently, IPM-based approaches have resulted in algorithms that solve linear programs approximately in \( \tilde{O}(n^\omega) \) arithmetic operations [6, 7]. We consider the linear programs of the form

\[
\min_{A^\top x = b, x \geq 0} c^\top x \quad \text{(primal)} \quad \text{and} \quad \max_{Ay \leq c} b^\top y \quad \text{(dual)},
\]

where \( A \in \mathbb{R}^{n \times d} \), \( b \in \mathbb{R}^d \), \( c \in \mathbb{R}^n \), and \( n \geq d \). Starting from a feasible solution, each iteration \( k \) of IPM corresponds to computing a vector of the following form

\[
\sqrt{W(k)} A (A^\top W(k) A)^{-1} A^\top \sqrt{W(k)} g(k), \quad (3.3.7)
\]

where \( W(k) \in \mathbb{R}^{n \times n} \) is a diagonal matrix and \( g \in \mathbb{R}^n \). Note that this is equivalent to solving a linear system with the matrix \( A^\top W(k) A \). Recent advances in IPMs [6, 7, 100] have shown that instead of (Equation 3.3.7), we can use the following vector

\[
\sqrt{\frac{1}{f} W(k)} A (A^\top \frac{1}{f} W(k) A)^{-1} A^\top \sqrt{\frac{1}{f} W(k)} g(k), \quad (3.3.8)
\]
where \( \mathbf{W}^{(k)} \in \mathbb{R}^{n \times n} \) is another diagonal matrix such that \( \| \mathbf{w}^{(k)} - \mathbf{\bar{w}}^{(k)} \|_{\infty} < C \) for some constant \( C \), and \( \mathbf{w}^{(k)} \) and \( \mathbf{\bar{w}}^{(k)} \) are the vectors corresponding to diagonal matrices \( \mathbf{W}^{(k)} \) and \( \mathbf{\bar{W}}^{(k)} \), respectively. Another insight from IPMs is that \( \mathbf{W}^{(k-1)} \) and \( \mathbf{W}^{(k)} \) are very close to each other in the sense that \( \| \mathbf{w}^{(k-1)} - \mathbf{w}^{(k)} \|_2 < \beta \) for some constant \( \beta \). Then the following lemma allows us to bound the number of low-rank changes we need to apply to \( \mathbf{\bar{W}}^{(k)} \) to maintain \( \| \mathbf{w}^{(k)} - \mathbf{\bar{w}}^{(k)} \|_{\infty} < C \) over the course of the algorithm. Therefore we can use the Sherman-Morrison-Woodbury identity (Fact 2.1.2.1) to maintain the inverse \( (\mathbf{A}^\top \mathbf{\bar{W}}^{(k)} \mathbf{A})^{-1} \), and this results in an algorithm for solving LPs with \( \tilde{O}(n^\omega) \) arithmetic operations.

In the original papers of Cohen-Lee-Song [6] and Brand [7], instead of the matrix \( (\mathbf{A}^\top \mathbf{\bar{W}}^{(k)} \mathbf{A})^{-1} \), the matrix \( \mathbf{A}(\mathbf{A}^\top \mathbf{\bar{W}}^{(k)} \mathbf{A})^{-1} \mathbf{A}^\top \) is maintained. The reason is that for a dense matrix (e.g., \( \text{NNZ} \mathbf{A} = \Omega(n^2) \)), the cost of multiplying \( \mathbf{A} \) by a vector in each iteration is \( \Omega(n^2) \). Therefore since the number of iterations of IPM is \( \sqrt{n} \), this alone gives a running time of \( \Omega(n^{2.5}) \), which is much higher than \( n^\omega \). However in our case, since \( \mathbf{A} \) is sparse with \( o(n^{\omega-1}) \) nonzero entries, the cost of this multiplication over the course of the algorithm is at most \( O(n^{\omega-0.5}) \). Therefore we focus on maintaining \( (\mathbf{A}^\top \mathbf{\bar{W}}^{(k)} \mathbf{A})^{-1} \).

To maintain \( (\mathbf{A}^\top \mathbf{\bar{W}}^{(k)} \mathbf{A})^{-1} \), we either have to use Fact 2.1.2.1 or compute \( (\mathbf{A}^\top \mathbf{\bar{W}}^{(k)} \mathbf{A})^{-1} \) from scratch. Consider a fix \( m \) for the sparse solver of [45] and the number of updates of rank \( \frac{n}{m} \) in the IPM, i.e., the number of indices \( k \) such that \( \frac{n}{m} \) entries are different between \( \mathbf{\bar{W}}^k \) and \( \mathbf{\bar{W}}^{k+1} \). By Lemma 2.4.2.6, the number of such changes is \( \tilde{O}(\sqrt{m}) \). If we recompute the inverse from scratch when we encounter these updates, then by (Equation 3.3.3), our cost is at lease

\[
\Omega(\text{NNZ} \mathbf{A} \cdot n \cdot m + n^\omega m^{2.5 - \omega}),
\]

which is larger than \( n^\omega \) because \( 2.5 > \omega \). If we use Sherman-Morrison Woodbury identity (Fact 2.1.2.1), since it is equivalent to applying the inverse to an \( n \times \frac{n}{m} \) matrix, the cost is at least

\[
\Omega(m^{0.5} \cdot s^\omega \cdot m^2) = \Omega(s^\omega \cdot m^{2.5}),
\]
because applying the inverse of $K^TAK$ to an $n \times \frac{n}{m}$ matrix costs at least $\Omega(s^\omega \cdot m^2)$. This is again more than $n^\omega$ because $ms = n$. Using the $\overline{XY}$ representation also leads to a cost of

$$\Omega(m^{1.5} \cdot \text{MM}(n,n,\frac{n}{m})),$$

which is again more than $n^\omega$. However, if based on our conjectures, we had a representation of the form $\overline{XX}^*$, then by (Equation 3.3.6), the cost of this would be

$$O(m^{0.5} \cdot \text{MM}(n,n,\frac{n}{m})),$$

which is smaller than matrix multiplication time. Now suppose our conjectures are true and we can find a representation of the inverse as $\overline{XX}^*$. To go below matrix multiplication time for this inverse maintenance problem, one can adapt the following approach: If the rank of the update is larger than $\frac{n}{m(\omega - 2/2)}$, recompute the inverse and $\overline{XX}^*$ from scratch. If the rank of the update is smaller than $n^\alpha$, use the Sherman-Morrison-Woodbury identity in an online way, i.e., compute the product of each term with the given vector separately (where $\alpha > 0.31$ is the dual of matrix multiplication exponent and is the largest number such that an $n \times n$ matrix can be multiplied with an $n \times n^\alpha$ matrix in $O(n^{2+o(1)})$ time). Finally if the rank of the update was between $n^\alpha$ and $\frac{n}{m(\omega - 2/2)}$, compute the update term of Sherman-Morrison-Woodbury identity (i.e., the second term) and store it as an explicit matrix $Q$.

With the above approach, the inverse operator is then given as $\overline{XX}^* + Q + M$, where $M$ is an implicit matrix given by Sherman-Morrison-Woodbury identity, i.e.,

$$M = -(\overline{XX}^* + Q)(\overline{A_S})^T(D^{-1} + \overline{A_S} \overline{XX}^* + Q)(\overline{A_S})^{-1} \overline{A_S} (\overline{XX}^* + Q),$$

where $S$ is the set of indices corresponding to updates to $\overline{W}$ that are not incorporated to $\overline{XX}^*$ or $Q$, and $D$ is the diagonal matrix corresponding to these updates. Note that the cost
of applying $Q$ to any matrix is the same as the cost of applying $\tilde{X}$.

Then one can see that by Lemma 2.4.2.6 and (Equation 3.3.6), the cost of inverse maintenance is bounded by

$$\tilde{O}(\text{NNZ}_A \cdot n \cdot m^{1.5+\omega/4} + n^{\omega} m^{0.5-\omega/4} + m^{(\omega-2)/4} \text{MM}(n, n, \frac{n}{m^{\omega-2/2}})),$$

(3.3.9)

where the first two terms come from the cost of recomputations of the inverse, and the other term comes from the updates performed using Sherman-Morrison-Woodbury identity. Now note that the exponent of $m$ in the second term of (Equation 3.3.9) is negative for any $\omega > 2$. For any value of $\text{NNZ}_A \ll n^{\omega-1}$, we can take $m$ small enough to make the first term less than $n^{\omega}$. Similarly, the third term is smaller than $n^{\omega}$. This can be checked by the online tool of [124].

In addition to inverse maintenance, one needs to consider the cost of queries for computing (Equation 3.3.8). For these, since we only need to have $\tilde{W}^{(k)}$ that is close to $W^{(k)}$, one does not need to compute all the entries of the formula. We only need to compute the entries that cause an entry of $\tilde{W}^{(k)}$ to change in the next iteration. This can be done by using heavy-hitters data structures in a way similar to their use in the recent works for solving tall dense linear programs, see [78, 79]. We omit the details of this here, but one can verify that with this approach, the total cost of queries can also be made less than matrix multiplication time. Therefore the overall approach gives an algorithm for solving linear programs with a sparse (i.e., $\text{NNZ}_A = o(n^{\omega-1})$) and poly-conditioned matrix faster than matrix multiplication time.

3.3.3 SoS decomposition of polynomials

If the coefficients of a degree $n$ polynomial $p$ is represented by a vector

$$\mathbf{a} = \begin{bmatrix} a_n & a_{n-1} & \cdots & a_1 & a_0 \end{bmatrix}^\top \in \mathbb{R}^{n+1},$$

168
then with \( x = \begin{bmatrix} x^n & x^{n-1} & \cdots & x & 1 \end{bmatrix}^\top \), \( c^\top x = p \). Another way of representing a polynomial \( p(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{2k} x^{2k} \) of even degree using a Hankel matrix is to define \( H \in \mathbb{R}^{(k+1) \times (k+1)} \) as

\[
H_{ij} = \begin{cases} a_{i+j} & \text{if } i+j \leq k+1, \\ a_{i+j-2} & \text{otherwise}. \end{cases}
\]

For example, for a degree 4 polynomial, we have

\[
H = \begin{bmatrix} a_0 & a_1 & a_2 \\ a_1 & a_2 & a_3 \\ a_2 & a_3 & a_4 \end{bmatrix}.
\]

Then one can see that with \( x = \begin{bmatrix} 1 & x & \cdots & x^{k-1} & x^k \end{bmatrix}^\top \), we have \( p = x^\top H x \). Now suppose there exists polynomials \( \ell_1, \ldots, \ell_m \) (of degree at most \( k \)) such that \( p = \sum_{j=1}^m \ell_j^2 \). Then showing the coefficient of \( \ell_j \) with \( b_0^{(j)}, \ldots, b_k^{(j)} \), for \( j \in [m] \), and defining the matrix \( B \in \mathbb{R}^{(k+1) \times m} \) as \( B_{r,j} = b_r^{(j)} \), we have \( p = x^\top B B^\top x \). Now note that a symmetric factorization of \( H \) like \( H = BB^\top \) gives us such coefficients for the polynomials. Moreover for \( j \in [m] \), we have

\[
\ell_j^2(x) = x^\top B_{:,j} B_{:,j}^\top x = x^\top \begin{bmatrix} b_0^{(j)} & b_1^{(j)} & \cdots & b_k^{(j)} \end{bmatrix}^\top \begin{bmatrix} b_0^{(j)} & b_1^{(j)} & \cdots & b_k^{(j)} \end{bmatrix} x
\]

\[
= (b_0^{(j)} + b_1^{(j)} x + \cdots + b_k^{(j)} x^k)^2.
\]

Therefore symmetric factorization of Hankel matrices give a sum-of-squares (SoS) decomposition of single-variable polynomials.

### 3.4 Symmetric Factorization of Hermitian Toeplitz Matrices

We start this section by showing how one can find a symmetric factorization of a certain rank-two Hermitian matrix. We will then use this to find a symmetric factorization for a Hermitian Toeplitz matrix.

169
Lemma 3.4.0.1. Let $M$ be a rank two Hermitian matrix of the following form.

$$M = \begin{bmatrix}
0 & \cdots & 0 & t_1^* & 0 & \cdots & 0 \\
0 & \cdots & 0 & t_2^* & 0 & \cdots & 0 \\
\vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots \\
0 & \cdots & 0 & t_{j-1}^* & 0 & \cdots & 0 \\
t_1 & \cdots & t_{j-1} & 0 & t_{j+1} & \cdots & t_n \\
0 & \cdots & 0 & t_{j+1}^* & 0 & \cdots & 0 \\
\vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots \\
0 & \cdots & 0 & t_n^* & 0 & \cdots & 0
\end{bmatrix}. \quad (3.4.1)$$

In other words, only the $j$‘th row and column of this matrix is nonzero and its entry $(j, j)$ is also zero. Then $M$ has exactly two nonzero eigenvalues $\lambda_1$ and $\lambda_2$ that are real and $\lambda_1 = -\lambda_2$. Moreover let $\tilde{v}_1$ and $\tilde{v}_2$ be the eigenvectors corresponding to $\lambda_1$ and $\lambda_2$, respectively. Also let $\lambda_1$ be the positive eigenvalue and $v_1 = \sqrt{\lambda_1}\tilde{v}_1$, $v_2 = \sqrt{\lambda_1}\tilde{v}_2$. Then $M = v_1v_1^* - v_2v_2^*$.

Proof. We calculate the eigenvalue decomposition of $M$. Since $M$ is Hermitian, its eigenvectors can be picked to be orthonormal, and since $M$ is a rank two matrix, it has at most two nonzero eigenvalues that can be computed by the formula $Mv = \lambda v$. This gives the following set of linear systems

$$\sum_{k \in [n], k \neq j} t_k v_k = \lambda v_j,$$

$$t_k^* v_j = \lambda v_k, \forall k \in [n], k \neq j. \quad (3.4.2)$$

Therefore for nonzero $\lambda$, we have $v_k = \frac{t_k^* v_j}{\lambda}$. Substituting this into the first equation, we have

$$\frac{v_j}{\lambda} \sum_{k \in [n], k \neq j} (t_k t_k^*) = \lambda v_j.$$  

Note that $v_1$ is nonzero because otherwise all of $v_k$‘s are zero by (Equation 3.4.2) (and
this is in contrast with the assumption that the norm of the eigenvectors is equal to one). Therefore

\[ \lambda^2 = \sum_{k \in [n], k \neq j}^n (t_k t_k^*) = \sum_{k \in [n], k \neq j} \| t_k \|^2. \]

Hence the right hand side is positive and \( \mathbf{M} \) has two real eigenvalues \( \lambda_1 = \sqrt{\sum_{k=1}^n (t_k t_k^*)} \) and \( \lambda_2 = -\sqrt{\sum_{k=1}^n (t_k t_k^*)} \), where we define \( t_j = 0 \). Let \( \mathbf{v}_1 \) and \( \mathbf{v}_2 \) be the eigenvectors corresponding to \( \lambda_1 \) and \( \lambda_2 \), respectively. Let \( \mathbf{v}_1 = \sqrt{\lambda_1} \mathbf{v}_1^* \) and \( \mathbf{v}_2 = \sqrt{\lambda_1} \mathbf{v}_2^* \). Note that since \( \lambda_1 \) is positive \( \sqrt{\lambda_1} \) is real and therefore \( \mathbf{v}_1^* = \sqrt{\lambda_1} \mathbf{v}_1^* \) and \( \mathbf{v}_2^* = \sqrt{\lambda_1} \mathbf{v}_2^* \). Then we have

\[ \mathbf{M} = \lambda_1 \mathbf{v}_1 \mathbf{v}_1^* + \lambda_2 \mathbf{v}_2 \mathbf{v}_2^* = \lambda_1 \mathbf{v}_1 \mathbf{v}_1^* - \lambda_1 \mathbf{v}_2 \mathbf{v}_2^* = \mathbf{v}_1 \mathbf{v}_1^* - \mathbf{v}_2 \mathbf{v}_2^*. \]

\[ \square \]

By Lemma 3.4.0.1, to find a symmetric factorization of \( \mathbf{M} \) in (Equation 3.4.1), we only need to find its eigenvalues and eigenvector. Since \( \mathbf{M} \) is a rank two matrix with \( O(n) \) nonzero entries, this can be done in \( O(n) \) time. To prove Theorem 3.2.0.3, we essentially find a symmetric factorization of such a matrix and show that a symmetric factorization of a Hermitian Toeplitz matrix can be constructed by shifting and adding this symmetric factorization for a rank two matrix.

**Theorem 3.2.0.3.** Let \( \mathbf{T} \in \mathbb{F}^{n \times n} \) be a Hermitian Toeplitz matrix with bit complexity \( \ell \). There exists an algorithm that finds a representation of matrices \( \mathbf{B} \) and \( \mathbf{C} \), each with \( n \) rows, \( O(n \log n) \) columns, and bit complexity \( \ell \) in time \( \tilde{O}(n \cdot \ell) \) such that \( \| \mathbf{T} - (\mathbf{B} \mathbf{B}^* - \mathbf{C} \mathbf{C}^*) \|_F < \frac{1}{2^\ell} \).

**Proof.** Let \( \mathbf{\tilde{T}} \) be a Toeplitz matrix that is equal to \( \mathbf{T} \) everywhere except on the diagonal and the diagonal of \( \mathbf{\tilde{T}} \) is equal to zero. We now show that \( \mathbf{\tilde{T}} \) can be written as \( \mathbf{V}_1 \mathbf{V}_1^* - \mathbf{V}_2 \mathbf{V}_2^* \) for \( \mathbf{V}_1, \mathbf{V}_2 \in \mathbb{F}^{n \times n} \).

Let \( \mathbf{M} = \mathbf{\tilde{T}} - \mathbf{\Delta} \mathbf{\tilde{T}} \mathbf{\Delta}^T \). Since \( \mathbf{\tilde{T}} \) is a Toeplitz matrix, \( \mathbf{M} \) is a matrix of rank two of the
following form

\[
M = \begin{bmatrix}
0 & t_2 & t_3 & \cdots & t_n \\
\vdots & \ddots & \ddots & \ddots & \vdots \\
t^*_1 & 0 & 0 & \cdots & 0 \\
0 & \ddots & \ddots & \ddots & \vdots \\
0 & \cdots & \ddots & \ddots & 0
\end{bmatrix}.
\]

Therefore by Lemma 3.4.0.1, there exists \( v_1 \) and \( v_2 \) such that \( M = v_1^*v_1^T - v_2^*v_2^T \). Now note that

\[
\tilde{T} = \sum_{j=1}^{n} \Delta^{j-1}M(\Delta^{j-1})^T.
\]

Therefore defining

\[
V_1 = \begin{bmatrix} v_1 & \Delta v_1 & \Delta^2 v_1 & \cdots & \Delta^{n-1} v_1 \end{bmatrix},
\]

\[
V_2 = \begin{bmatrix} v_2 & \Delta v_2 & \Delta^2 v_2 & \cdots & \Delta^{n-1} v_2 \end{bmatrix},
\]

we have \( \tilde{T} = V_1 V_1^* - V_2 V_2^* \). Let \( t_1 \) be the diagonal element of \( T \) that is a real number because \( T \) is a Hermitian matrix. Then if \( t_1 \geq 0 \), setting

\[
B = \begin{bmatrix} V_1 & \sqrt{|t_1|}I \end{bmatrix}, \text{ and } C = V_2,
\]

and setting

\[
B = V_1, \text{ and } C = \begin{bmatrix} V_2 & \sqrt{|t_1|}I \end{bmatrix},
\]

otherwise, we have \( T = BB^* - CC^* \). Finally note that in the first case \( C \) is a Toeplitz matrix, and therefore has a displacement rank of two and \( B \) has a displacement rank of two with respect to \( (\Delta, \begin{bmatrix} \Delta & \Delta \end{bmatrix}^T) \). Similarly, in the second case also the displacement rank of both matrices is two. This implies that a vector can be multiplied by \( B, C, B^*, C^* \) in \( \tilde{O}(n) \) time by FFT techniques. □
3.5 Key Identity for Hankel Matrices

In this section, we consider a symmetric factorization of a Hankel matrix $H$ and without loss of generality, we assume $H \in \mathbb{F}^{2^k \times 2^k}$, for $k \in \mathbb{N}$. Note that if the dimensions of $H$ is not a power of two, we can extend it to a Hankel matrix $\tilde{H}$ in which the dimensions are a power of two as the following. Let $h = (h_1, \ldots, h_{2s-1})$, for $s \in \mathbb{N}$, be the generating vector of the Hankel matrix $H$. In this case $H \in \mathbb{F}^{s \times s}$, and we are assuming $s$ is not a power of two. Let $k$ be the smallest integer such that $2^k > s$ and let $\tilde{h} = (h_1, \ldots, h_{2s-1}, 0, \ldots, 0) \in \mathbb{F}^{2^k+1-1}$. Now let $\tilde{H}$ be a Hankel matrix with generating vector $\tilde{h}$. Then $\tilde{H} \in \mathbb{F}^{2^k \times 2^k}$. Moreover $\tilde{H}_{1:s, 1:s} = H$. Therefore if $\tilde{B}, \tilde{C}$ are matrices such that $\tilde{H} = \tilde{B}\tilde{B}^* - \tilde{C}\tilde{C}^*$, then defining $B = \tilde{B}_{1:s,:}$ and $C = \tilde{C}_{1:s,:}$, we have $H = BB^* - CC^*$. Therefore we only need to find a symmetric factorization of $\tilde{H}$. We now define a matrix that converts a Hankel matrix to Toeplitz and vice versa.

Definition 3.5.0.1. Let $J_n \in \mathbb{F}^{n \times n}$ be a matrix with $J_n(i,j) = 1$ if $i + j = n + 1$, and $J_n(i,j) = 0$, otherwise. For example

$$J_4 = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}.$$  

We call this matrix the exchange matrix (also called backward identity). When the dimension is clear from the context, we show the exchange matrix with just $J$. Note that $JJ = I$, and $J^T = J$. Moreover we say, a matrix $M$ is centrosymmetric if $JM = MJ$, is persymmetric if $MJ = JM^T$, and is bisymmetric if it is both symmetric and centrosymmetric.

The next lemma describes our similarity transformation to decompose a Hankel matrix to the sum of a Hermitian Toeplitz matrix and a centrosymmetric Hankel matrix.
Lemma 3.5.0.2 (Key Identity). Let \( S = \frac{1}{2}(1+i)I + \frac{1}{2}(1-i)J \), where \( I \) is the identity matrix and \( J \) is the exchange matrix. Let \( H \) be a Hankel matrix. The imaginary part of \( SHS^* \) is a skew-symmetric Toeplitz matrix with zero diagonal and its real part is a centrosymmetric Hankel matrix.

Proof. We have

\[
SHS^* = \left( \frac{1}{2}(1+i)I + \frac{1}{2}(1-i)J \right) H \left( \frac{1}{2}(1+i)I + \frac{1}{2}(1-i)J \right)^*
\]

\[
= \left( \frac{1}{2}(1+i)I + \frac{1}{2}(1-i)J \right) H \left( \frac{1}{2}(1-i)I + \frac{1}{2}(1+i)J \right)
\]

\[
= \frac{1}{4} \left( (1+i)(1-i)H + (1+i)(1-i)JH + (1+i)^2HJ + (1-i)^2JH \right)
\]

\[
= \frac{1}{2}(H + JHJ) + \frac{i}{2}(HJ - JH).
\]

Therefore the real part of \( SHS^* \) is \( \frac{1}{2}(H + JHJ) \). Now we have

\[(H + JHJ)J = HJ + JH = J(JHJ + H).\]

Therefore \( \text{real}(SHS^*) \) is centrosymmetric. Also note that both \( H \) and \( JHJ \) are Hankel and the sum of Hankel matrices is a Hankel matrix. Therefore \( \text{real}(SHS^*) \) is also Hankel. In addition, note that since \( H \) is Hankel, both \( HJ \) and \( JH \) are Toeplitz matrices and the sum (and also the difference) of Toeplitz matrices, is a Toeplitz matrix. Therefore \( HJ - JH \) and \( \text{imag}(SHS^*) \) are Toeplitz matrices. Finally we have


Therefore \( \text{imag}(SHS^*) \) is a skew-symmetric matrix, and hence its diagonal is equal to zero. Note that this implies \( i \cdot \text{imag}(SHS^*) \) is a Hermitian matrix.

\[\Box\]
We now give a 4-by-4 example to understand Lemma 3.5.0.2 better. Let

\[
H = \begin{bmatrix}
h_1 & h_2 & h_3 & h_4 \\
h_2 & h_3 & h_4 & h_5 \\
h_3 & h_4 & h_5 & h_6 \\
h_4 & h_5 & h_6 & h_7 \\
\end{bmatrix}.
\]

We then have

\[
\text{real}(SHS^*) = \frac{1}{2} \begin{bmatrix}
h_1 + h_7 & h_2 + h_6 & h_3 + h_5 & 2h_4 \\
h_2 + h_6 & h_3 + h_5 & 2h_4 & h_2 + h_6 \\
h_3 + h_5 & 2h_4 & h_3 + h_5 & h_2 + h_6 \\
2h_4 & h_3 + h_5 & h_2 + h_6 & h_1 + h_7 \\
\end{bmatrix},
\]

\[
\text{imag}(SHS^*) = \frac{1}{2} \begin{bmatrix}
0 & h_3 - h_5 & h_2 - h_6 & h_1 - h_7 \\
h_5 - h_3 & 0 & h_3 - h_5 & h_2 - h_6 \\
h_6 - h_2 & h_5 - h_3 & 0 & h_3 - h_5 \\
h_7 - h_1 & h_6 - h_2 & h_5 - h_3 & 0 \\
\end{bmatrix}.
\]

Now note that matrix $S$ is a unitary matrix and therefore $SS^* = S^*S = I$. Therefore, we have

\[
H = S^*SHS^* = S^* \left( \text{real}(SHS^*) + i \cdot \text{imag}(SHS^*) \right) S
\]  

(3.5.1)

Therefore if we have matrices $B_1, B_2, C_1, C_2$ such that $\text{real}(SHS^*) = B_1B_1^* - C_1C_1^*$ and $i \cdot \text{imag}(SHS^*) = B_2B_2^* - C_2C_2^*$, then we have

\[
H = \begin{bmatrix}
S^*B_1 & S^*B_2 \\
S^*B_1 & S^*B_2 \\
\end{bmatrix}^* - \begin{bmatrix}
S^*C_1 & S^*C_2 \\
S^*C_1 & S^*C_2 \\
\end{bmatrix}^*.
\]

In the next section, we discuss how Lemma 3.5.0.2 can be exploited to devise our recursive


Algorithm 14: Symmetric Factorization of Hankel Matrices

1. **Input:** Hankel matrix $H \in \mathbb{R}^{2^k \times 2^k}$
2. Set $H_0 = H$
3. for $t = 1, \ldots, k$ do
   4. Set $N_t = i \cdot \text{imag}(\overline{S}_t H_{t-1} S_t^*)$
   5. Set $H_t = \text{real}(\overline{S}_t H_{t-1} S_t^*)$
   6. Set $N_{k+1} = H_k$
4. for $t = 1, \ldots, k+1$ do
   5. Set $X_t$ and $Y_t$ to be matrices such that $N_t = X_t X_t^* - Y_t Y_t^*$
   6. Set $B_t = \overline{S}_1^* S_2^* \cdots \overline{S}_{t-1}^* X_t$ and $C_t = \overline{S}_1^* S_2^* \cdots \overline{S}_{t-1}^* Y_t$
10. return $[B_{k+1} B_k \cdots B_2 B_1]$ and $[C_{k+1} C_k \cdots C_2 C_1]$

3.6 Symmetric Factorization of Hankel Matrices

Since $i \cdot \text{imag}(SHS^*)$ in (Equation 3.5.1) is a Hermitian Toeplitz matrix, we can use Theorem 3.2.0.3 to find a symmetric factorization for it. To deal with the real part of $SHS^*$, we use Lemma 3.5.0.2 in a recursive fashion using the following matrix.

**Definition 3.6.0.1.** For $n = 2^k$, and $t = 1, \ldots, k$, we define $S_t := \frac{1}{2} (1 + i) I_{n/2^{t-1}} + \frac{1}{2} (1 - i) J_{n/2^{t-1}} \in \mathbb{F}^{(2^{k-t+1}) \times (2^{k-t+1})}$, and

$$
\overline{S}_t := \begin{bmatrix}
S_t & 0 & \cdots & 0 \\
0 & S_t & \cdots & 0 \\
\vdots & \vdots & \ddots & \vdots \\
0 & 0 & \cdots & S_t
\end{bmatrix} \in \mathbb{R}^{2^k \times 2^k}.
$$

Algorithm 14 is our main procedure to find a symmetric factorization of a Hankel matrix. However, so far, we only know how to find a symmetric factorization of $N_1$ using Theorem 3.2.0.3. Therefore we discuss how to find a symmetric factorization for the rest of $N_t$’s. We start by characterizing the structure of matrices $H_t$ and $N_t$. 

176
Lemma 3.6.0.2. For \( t = 1, \ldots, k \), matrix \( H_t \) consists of \( 2^t \times 2^t \) blocks of Hankel matrices as the following. The first block-row consists of \( 2^t \) Hankel matrices

\[
\begin{bmatrix}
Q_1 & Q_2 & \cdots & Q_{2^t-1} & Q_{2^t}
\end{bmatrix}.
\]

The second block-row is

\[
\begin{bmatrix}
Q_2 & JQ_1J & Q_4 & JQ_3J & \cdots & Q_{2^t} & JQ_{2^t-1}J
\end{bmatrix}
\]

For \( s = 1, \ldots, t - 1 \), let \( P_s \) be the matrix consisting of block-rows \( 1 \) to \( 2^s \), and \( \overline{P}_s \) be the matrix consisting of block-rows \( 2^s + 1 \) to \( 2^{s+1} \). Let

\[
P_s = \begin{bmatrix}
P_{s,1} & P_{s,2} & \cdots & P_{s,2^s-s}
\end{bmatrix}.
\]

Then

\[
\overline{P}_s = \begin{bmatrix}
P_{s,2} & P_{s,1} & P_{s,4} & P_{s,3} & \cdots & P_{s,2^s-s} & P_{s,2^s-s-1}
\end{bmatrix}.
\]

Moreover the structure of the block-columns of \( H_t \) is similar to the structure of the block-rows we described above.

Proof. Before proving the lemma, note that the description completely describes \( H_t \) (at least up to the block structure) since it describes the first two block-rows and then it uses the first \( 2^s \) block-rows to describe the next \( 2^s \) block rows. The structure of \( H_1 \) follows from Lemma 3.5.0.2. We then use induction to prove the structure for the rest of \( H_t \)'s.

First note that the number of block-rows and block-columns of \( H_t \) is twice the number of block-rows and block-columns of \( H_{t-1} \). In other words, each block of \( H_{t-1} \) is split into four blocks in \( H_t \). Note that in iteration \( t \), we multiply \( H_{t-1} \) by \( S_t \) and \( S_t^* \) from left and right, respectively, which is equivalent to multiplying each block of \( H_{t-1} \) by \( S_t \) and \( S_t^* \) from left and right, respectively. Therefore by induction hypothesis for \( H_{t-1} \) and Lemma 3.5.0.2, since the blocks of \( H_{t-1} \) are Hankel matrices, the blocks of \( H_t = \text{real}(S_t H_{t-1} S_t^*) \) are also
Hankel matrices. Moreover the relation between the first block-row and the second block-row directly follows from Lemma 3.5.0.2 due to centrosymmetry of the resulting Hankel matrix. Finally the relation between $P_s$ and $\bar{P}_s$ for $s = 1, \ldots, t - 1$ simply follows from the induction hypothesis for the structure of $H_{t-1}$. A similar argument proves the structure of block-columns as well.

□

We now use Lemma 3.6.0.2 to characterize the structure of matrices $N_t$.

**Lemma 3.6.0.3.** For $t = 1, \ldots, k$, matrix $N_t$ consists of $2^{t-1} \times 2^{t-1}$ blocks of Hermitian Toeplitz matrices as the following. The first block-row consists of $2^{t-1}$ Toeplitz matrices

$$\begin{bmatrix} Q_1 & Q_2 & \cdots & Q_{2^{t-1}} & Q_{2^{t-1}} \end{bmatrix}.$$  

The second block-row is

$$\begin{bmatrix} Q_2 & JQ_1J & Q_4 & JQ_3J & \cdots & Q_{2^{t-1}} & JQ_{2^{t-1}-1}J \end{bmatrix}$$

For $s = 1, \ldots, t - 2$, let $P_s$ be the matrix consisting of block-rows $1$ to $2^s$, and $\bar{P}_s$ be the matrix consisting of block-rows $2^s + 1$ to $2^{s+1}$. Let

$$P_s = \begin{bmatrix} P_{s,1} & P_{s,2} & \cdots & P_{s,2^{t-s}} \end{bmatrix}.$$  

Then

$$\bar{P}_s = \begin{bmatrix} P_{s,2} & P_{s,1} & P_{s,4} & P_{s,3} & \cdots & P_{s,2^{t-s}} & P_{s,2^{t-s}-1} \end{bmatrix}.$$ 

Moreover for even $j$, $Q_j = 0$. Also the structure of the block-columns of $N_t$ is similar to the structure of the block-rows we described above.

**Proof.** Note that if a real matrix $M$ is skew-symmetric, $i \cdot M$ is Hermitian. Therefore the structure of $N_1$ follows from Lemma 3.5.0.2. For $t = 2, \ldots, k$, note that $N_t = i \cdot \text{imag}(\bar{S}_t H_{t-1}^* \bar{S}_t^*)$. Therefore we use the structure of $H_{t-1}$ described in Lemma 3.6.0.2 to
prove the structure for \( N_t \). Note that the number of blocks of \( N_t \) is equal to the number of blocks of \( H_{t-1} \). Moreover for each block \( Q \) of \( H_{t-1} \), the corresponding block in \( N_t \) is \( i \cdot \text{imag}(S_tQS_t^*) \), which is a Hermitian Toeplitz matrix by Lemma 3.5.0.2. Moreover the structure of block-rows of \( N_t \) (i.e., the relation between the first block-row and the second block-row and the relation between \( P_s \) and \( \tilde{P}_s \), for \( s = 1, \ldots, t - 2 \)) follows from the structure of \( H_{t-1} \) due to Lemma 3.6.0.2 and the fact that \( J \) and \( S_t \) commute, i.e., \( JS_t = S_tJ \). The structure of block columns also follows similarly.

Finally note that if \( Q \) is a centrosymmetric Hankel matrix, then \( \text{imag}(S_tQS_t^*) \) is zero since

\[
\text{imag}(S_tQS_t^*) = \frac{1}{2}(QJ - JQ) = 0,
\]

where the second equality follows from the definition of centrosymmetry (see Definition 3.5.0.1). For the first equality see the proof of Lemma 3.5.0.2. Finally note that for a Hankel matrix \( Q \), by Lemma 3.5.0.2, \( \text{real}(S_tQS_t^*) \) is a centrosymmetric Hankel matrix. Therefore the top-right and bottom-left blocks of \( \text{real}(S_tQS_t^*) \) are also centrosymmetric Hankel matrices. Therefore since the blocks of \( H_{t-1} \) are Hankel matrices, the blocks \( Q_j \) with even index \( j \) in \( H_{t-1} \) are centrosymmetric Hankel and therefore the corresponding blocks of them in \( N_t \) are zero. \( \square \)

Before going further, we need to define the following matrices that allow us to exploit the structures described in Lemmas 3.6.0.2 and 3.6.0.3.

**Definition 3.6.0.4.** For \( t \in [k] \), let

\[
E_t = \begin{bmatrix}
0_{2^{t-1} \times 2^{t-1}} & 0_{2^{t-1} \times (2^k - 2^{t-1})} \\
0_{(2^k - 2^{t-1}) \times 2^{t-1}} & I_{2^k - 2^{t-1}}
\end{bmatrix} \in \mathbb{R}^{2^k \times 2^k},
\]
For example, the matrix $F_2$ allows us to permute $P_1$ in $H_k$ to get $\tilde{P}_1$ in Lemma 3.6.0.2. We then can remove the first entry of $P_1$ using $E_2$ to prevent it with clashing with the first column. Note that $E_i^T = E_i$ and $F_i^T = F_i$. Therefore we can use these matrices for permuting block-columns as well. For the other block-rows/columns, we can use appropriate $F_i$'s and $E_i$'s. We use these matrices in the proofs of the rest of the section.

We are now equipped to describe how a representation of symmetric factorization of matrices $N_t$, for $t \in [k]$, can be found in linear time.

**Theorem 3.6.0.5.** For $t = 1, \ldots, k$, we can find $X_t$ and $Y_t$ in $\tilde{O}(n)$ time such that $N_t = X_tX_t^* - Y_tY_t^*$.

**Proof.** Let $\begin{bmatrix} Q_1 & \cdots & Q_{2^{r-1}} \end{bmatrix}$ be the blocks of the first block-row and the first block-column of $N_t$, respectively. For $s \in [t]$, let $M_s$ be a matrix with block structure as $N_t$ such that all of its blocks are zero except the block-rows/columns $1, \ldots, 2^{s-1}$ and its block-rows/columns $1, \ldots, 2^{s-1}$ are equal to the block-rows/columns $1, \ldots, 2^{s-1}$ of $N_t$. For example $M_t = N_t$

$$
M_t = \begin{bmatrix}
Q_1 & Q_2 & Q_3 & \cdots & Q_{2^{r-1}} \\
Q_2 & 0 & 0 & \cdots & 0 \\
Q_3 & 0 & 0 & \cdots & 0 \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
Q_{2^{r-1}} & 0 & 0 & \cdots & 0
\end{bmatrix}.
$$
For $s \in [2^{t-1}]$, let $Q_{s,U}$ be an upper triangular matrix that is equal to $Q_s$ on the diagonal and above the diagonal, and let $Q_{s,L}$ be a lower triangular matrix with zero diagonal that is equal to $Q_s$ below the diagonal. Therefore $Q_s = Q_{s,U} + Q_{s,L}$. Moreover since $Q_s$ is Hermitian by Lemma 3.6.0.3, $Q_s = Q_{s,U}^* + Q_{s,L}^*.$

Let $U$ and $L$ be matrices with the same block structure as $N_t$. Moreover suppose all of blocks of $U$ and $L$ are zero except the first block-row and the first block-column. Let the first block-row and the first block-column of $U$ be

$$\begin{bmatrix} \frac{1}{2}Q_1 & Q_{2,U} & Q_{3,U} \cdots Q_{2^{t-1},U} \end{bmatrix}, \text{ and } \begin{bmatrix} \frac{1}{2}Q_1 & Q_{2,U} & Q_{3,U} \cdots Q_{2^{t-1},U} \end{bmatrix}^*,$$

respectively. Also let the first block-row and the first block-column of $L$ be

$$\begin{bmatrix} \frac{1}{2}Q_1 & Q_{2,L} & Q_{3,L} \cdots Q_{2^{t-1},L} \end{bmatrix}, \text{ and } \begin{bmatrix} \frac{1}{2}Q_1 & Q_{2,L} & Q_{3,L} \cdots Q_{2^{t-1},L} \end{bmatrix}^*,$$

respectively. Therefore $M_1 = U + L$. We now give symmetric factorizations for $U$ and $L$. By construction and Lemma 3.5.0.2, the diagonal of $Q_1$ is zero. Therefore by Lemma 3.4.0.1, the matrix consisting of only the first row and the first column of $U$ can be written as $v_1^* - v_2^*$ for some vectors $v_1$ and $v_2$ that can be computed in
$O(n)$ time. Now since $Q_1$ is Toeplitz and $Q_{s,U}$'s are upper triangular and Toeplitz, we have

$$U = \sum_{j=0}^{2k-t+1-1} \tilde{\Lambda}^j (U - \tilde{\Lambda}U\tilde{\Lambda}^\top)(\tilde{\Lambda}^j)^\top.$$  

Therefore setting

$$V_1 = \begin{bmatrix} v_1 & \tilde{\Lambda}v_1 & \tilde{\Lambda}^2v_1 & \cdots & \tilde{\Lambda}^{2k-t+1-1}v_1 \end{bmatrix},$$

$$V_2 = \begin{bmatrix} v_2 & \tilde{\Lambda}v_2 & \tilde{\Lambda}^2v_2 & \cdots & \tilde{\Lambda}^{2k-t+1-1}v_2 \end{bmatrix},$$

we have $U = V_1 V_1^* - V_2 V_2^*$. Now consider the matrix that is equal to zero everywhere except on row $2k-t+1$ and column $2k-t+1$ and on that row and column, it is equal to $L$. This matrix is equal to

$$L - \tilde{\Lambda}^\top L\tilde{\Lambda}.$$  

Because the diagonal of $Q_1$ is equal to zero, by Lemma 3.4.0.1, we can find $w_1$ and $w_2$ such that $L - \tilde{\Lambda}^\top L\tilde{\Lambda} = w_1 w_1^* - w_2 w_2^*$ in $O(n)$ time. Now since $Q_1$ is Toeplitz and $Q_{s,L}$'s are lower triangular and Toeplitz, we have

$$L = \sum_{j=0}^{2k-t+1-1} (\tilde{\Lambda}^j)^\top (L - \tilde{\Lambda}^\top L\tilde{\Lambda})\tilde{\Lambda}^j.$$  

Therefore setting

$$W_1 = \begin{bmatrix} w_1 & \tilde{\Lambda}^\top w_1 & (\tilde{\Lambda}^2)^\top w_1 & \cdots & (\tilde{\Lambda}^{2k-t+1-1})^\top w_1 \end{bmatrix},$$

$$W_2 = \begin{bmatrix} w_2 & \tilde{\Lambda}^\top w_2 & (\tilde{\Lambda}^2)^\top w_2 & \cdots & (\tilde{\Lambda}^{2k-t+1-1})^\top w_2 \end{bmatrix},$$

we have $L = W_1 W_1^* - W_2 W_2^*$. Hence

$$M_1 = \begin{bmatrix} V_1 & W_1 \end{bmatrix} \begin{bmatrix} V_1 & W_1 \end{bmatrix}^* - \begin{bmatrix} V_2 & W_2 \end{bmatrix} \begin{bmatrix} V_2 & W_2 \end{bmatrix}^*.$$
We now construct other $M_s$'s recursively. By using matrices in Definition 3.6.0.4 and the structure of $N_t$ described in Lemma 3.6.0.3, for $s = 2, \ldots, t$, we have

$$M_s = M_{s-1} + E_{k-t+s+1}F_{k-t+s+1}M_{s-1}F_{k-t+s+1}^T + E_{k-t+s+1}^T$$

Therefore if $P_s$ and $G_s$ are matrices such that $M_s = P_sP_s^* - G_sG_s^*$, then for $s = 2, \ldots, t$

$$P_s = \begin{bmatrix} P_{s-1} & E_{k-t+s+1}F_{k-t+s+1}P_{s-1} \\ G_{s-1} & E_{k-t+s+1}F_{k-t+s+1}G_{s-1} \end{bmatrix},$$

$$G_s = \begin{bmatrix} P_{s-1} & E_{k-t+s+1}F_{k-t+s+1}G_{s-1} \\ G_{s-1} & E_{k-t+s+1}F_{k-t+s+1}^T \end{bmatrix}.$$

This completes the proof since $M_t = N_t$. □

We now prove the main theorem. In addition to Theorem 3.6.0.5, this only requires describing how to find a representation of symmetric factorization of $N_{k+1} = H_k$ in linear time.

*Proof of Theorem 3.2.0.2.* By Theorem 3.6.0.5, we can find $X_t$ and $Y_t$ such that $N_t = X_tX_t^* - Y_tY_t^*$, for $t \in [k]$. Therefore we only need to find $X_{k+1}$ and $Y_{k+1}$ such that $N_{k+1} = X_{k+1}X_{k+1}^* - Y_{k+1}Y_{k+1}^*$. Note that $N_{k+1} = H_k$.

The matrix $H_k$ is $2^k$-by-$2^k$ and according to Lemma 3.6.0.2, it consists of $2^k$-by-$2^k$ blocks. Therefore each block of it is only one entry. In this case for a block $Q$, we have $Q = QJQJ^T$. Therefore all of the entries on the diagonal of $H_k$ are the same. Moreover, each row is just a permutation of the first row and similarly each column is a permutation of the first column. Now let $H_k$ be the matrix that is equal to $H_k$ everywhere except on the diagonal and $H_k$ is zero on the diagonal, i.e., $H_k = H_k + H_k(1,1) \cdot I$.

Now let $M_0$ be a matrix with all of the entries equal to zero except the first row and the first column and its first row and column is equal to the first row and column of $H_k$. By Lemma 3.4.0.1, we can find vectors $v_1$ and $v_2$ such that $M_0 = v_1v_1^* - v_2v_2^*$ in $O(n)$ time.

Now for $t \in [k]$, let $M_t$ be the matrix that is zero everywhere except on rows/columns
1, \ldots, 2^t and its rows/columns 1, \ldots, 2^t are equal to the corresponding rows/columns of \( \mathbf{H}_k \). Note that \( \mathbf{M}_k = \mathbf{H}_k \). Now because of the structure of \( \mathbf{H}_k \) described by Lemma 3.6.0.2, for \( t \in [k] \), we have

\[
\mathbf{M}_t = \mathbf{M}_{t-1} + \mathbf{E}_t \mathbf{F}_t \mathbf{M}_{t-1} \mathbf{F}_t^\top \mathbf{E}_t^\top
\]  

(3.6.1)

Let \( \mathbf{P}_t \) and \( \mathbf{G}_t \) be such that \( \mathbf{M}_t = \mathbf{P}_t \mathbf{P}_t^* - \mathbf{G}_t \mathbf{G}_t^* \). Then by (Equation 3.6.1), we have

\[
\mathbf{M}_t = \begin{bmatrix} \mathbf{P}_{t-1} & \mathbf{E}_{t-1} \mathbf{F}_{t-1} \mathbf{P}_{t-1} \\ \mathbf{E}_{t-1} \mathbf{F}_{t-1} \mathbf{P}_{t-1} & \mathbf{P}_{t-1} \mathbf{E}_{t-1} \mathbf{F}_{t-1} \mathbf{P}_{t-1} \end{bmatrix}^*
\]

- \begin{bmatrix} \mathbf{G}_{t-1} & \mathbf{E}_{t-1} \mathbf{F}_{t-1} \mathbf{G}_{t-1} \\ \mathbf{E}_{t-1} \mathbf{F}_{t-1} \mathbf{G}_{t-1} & \mathbf{G}_{t-1} \end{bmatrix}^*.

Therefore \( \mathbf{P}_t = \begin{bmatrix} \mathbf{P}_{t-1} & \mathbf{E}_{t-1} \mathbf{F}_{t-1} \mathbf{P}_{t-1} \end{bmatrix} \) and \( \mathbf{G}_t = \begin{bmatrix} \mathbf{G}_{t-1} & \mathbf{E}_{t-1} \mathbf{F}_{t-1} \mathbf{G}_{t-1} \end{bmatrix} \). Therefore, we only need to find \( \mathbf{v}_1 \) and \( \mathbf{v}_2 \) with \( \mathbf{M}_0 = \mathbf{v}_1 \mathbf{v}_1^* - \mathbf{v}_2 \mathbf{v}_2^* \) to completely describe \( \mathbf{H}_k \). Now if \( H_k(1, 1) \geq 0 \), we set

\[
\mathbf{X}_{k+1} = \begin{bmatrix} \mathbf{P}_k & \sqrt{\mathbf{H}_k(1, 1)} \cdot \mathbf{I} \end{bmatrix}, \quad \text{and} \quad \mathbf{Y}_{k+1} = \mathbf{G}_k,
\]

and we set

\[
\mathbf{X}_{k+1} = \mathbf{P}_k, \quad \text{and} \quad \mathbf{Y}_{k+1} = \begin{bmatrix} \mathbf{G}_k & \sqrt{\mathbf{H}_k(1, 1)} \cdot \mathbf{I} \end{bmatrix},
\]

otherwise. Therefore we can also find a symmetric factorization of \( \mathbf{N}_{k+1} \) in \( \tilde{O}(n) \) time and this combined with Theorem 3.6.0.5 completes the proof.

\[\square\]

### 3.7 Symmetric Factorization of Inverses of Hankel Matrices

In this section, we prove our result for symmetric factorizations of the inverses of Hankel matrices (Theorem 3.2.0.5). We actually prove a more general result: for a given matrix \( \mathbf{M} \in \mathbb{R}^{n \times n} \) with Sylvester-type displacement rank of two and bit complexity \( \ell \), we show how to find a representation of the matrices \( \mathbf{B} \) and \( \mathbf{C} \) with \( n \) rows, \( \tilde{O}(n) \) columns, and
bit complexity $\ell$ in time $\tilde{O}(n^{\omega/2} \cdot \ell)$ such that $\| M - (BB^* - CC^*) \|_F < \frac{1}{2^r}$. Then since the inverse of a Hankel matrix has a Sylvester-type displacement rank of two, this gives an algorithm for the inverse of a Hankel matrix. A key technique in our algorithm is the following lemma which is similar to Lemma 3.5.0.2 with the important difference that when we apply the recursion arising from this lemma, the displacement rank of the matrix doubles (instead of staying the same). This then forces us to stop the recursion when the size of the blocks is $\sqrt{n}$.

**Lemma 3.7.0.1.** Let $S = \frac{1}{2} (1 + i)I + \frac{1}{2} (1 - i)J$, where $I$ is the identity matrix and $J$ is the exchange matrix (see Definition 3.5.0.1). Let $M$ be an $n \times n$ real symmetric matrix with Sylvester-type displacement rank of less than or equal to $r$ with respect to $(\Delta, \Delta^T)$ and $(\Delta^T, \Delta)$. Then $\text{real}(SM^*)$ is bisymmetric and has a Sylvester-type displacement rank of at most $2r$ with respect to $(\Delta, \Delta^T)$ and $(\Delta^T, \Delta)$. Moreover $\text{imag}(SM^*)$ is persymmetric Hermitian and has a Stein-type displacement rank of at most $2r + 2$ with respect to $(\Delta, \Delta^T)$. Also the diagonal entries of $\text{imag}(SM^*)$ are zero.

**Proof.** We first write each part of $SM^*$. We have

$$SM^* = \left( \frac{1}{2} (1 + i)I + \frac{1}{2} (1 - i)J \right) M \left( \frac{1}{2} (1 - i)I + \frac{1}{2} (1 + i)J \right)$$

$$= \frac{1}{2} (M + JMJ) + \frac{i}{2} (MJ - JM).$$

Note that $M + JMJ$ is symmetric because both $M$ and $JMJ$ are symmetric. Moreover

$$(M + JMJ)J = MJ + JM = J(JMJ + M).$$

Therefore $\text{real}(SM^*) = \frac{1}{2} (M + JMJ)$ is bysymmetric. Now we show it has a Sylvester-type displacement rank of at most $2r$ with respect to $(\Delta, \Delta^T)$. We need to show that

$$\text{rank}(\Delta(M + JMJ) - (M + JMJ)\Delta^T) \leq 2r.$$
One can easily verify that $\Delta J = J\Delta^T$. Therefore

$$\Delta JM J - JM J\Delta^T = J(\Delta^T M - M\Delta)J.$$ 

Thus since $J$ is a full rank matrix,

$$\text{rank}(\Delta(M + JM J) - (M + JM J)\Delta^T) \leq \text{rank}(\Delta M - M\Delta^T) + \text{rank}(\Delta JM J - JM J\Delta^T)$$

$$= \text{rank}(\Delta M - M\Delta^T) + \text{rank}(J(\Delta^T M - M\Delta)J)$$

$$= \text{rank}(\Delta M - M\Delta^T) + \text{rank}(\Delta^T M - M\Delta)$$

$$\leq 2r.$$ 

We can bound the Sylvester-type displacement rank of real$(SMS^*)$ with respect to $(\Delta^T, \Delta)$ in a similar way. Now we turn to the imaginary part. Since $M$ is real and symmetric $M^* = M$. Hence we have

$$(i(MJ - JM))^* = -i(JM^* - M^*J) = -i(JM - MJ) = i(MJ - JM).$$

Therefore $\text{imag}(SMS^*)$ is Hermitian. Again since $M$ is real and symmetric $M^T = M$. Therefore

$$i(MJ - JM)J = i(M - JM J) = iJ(JM - MJ) = iJ(M^T J - JM^T)^T = iJ(MJ - JM)^T.$$ 

Now we need to show that $\text{rank}((MJ - JM) - (M - JM)\Delta^T) \leq 2r$. We have

$$MJ - \Delta M J\Delta^T = (M - \Delta M\Delta)J.$$ 

Therefore

$$\text{rank}(MJ - \Delta M J\Delta^T) = \text{rank}(M - \Delta M\Delta).$$

Let $\Delta$ be a matrix obtained from $\Delta$ by changing the $(1, n)$ entry from zero to one. For
Therefore $\tilde{\Delta}$ is full-rank. Moreover let $\tilde{\mathbf{I}}$ be a matrix obtained by $\mathbf{I}$ by changing the $(1,1)$ entry from one to zero. For example

$$
\tilde{\mathbf{I}} = \begin{bmatrix}
0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 \\
\end{bmatrix}.
$$

Then we have $\tilde{\Delta}^T \Delta = \tilde{\mathbf{I}}$. Since $\tilde{\Delta}$ is full rank,

$$
\text{rank}(\mathbf{M} - \Delta \mathbf{M} \Delta) = \text{rank}(\tilde{\Delta}^T \mathbf{M} - \tilde{\mathbf{I}} \mathbf{M} \Delta)
$$

Now note that $\tilde{\Delta}$ differ from $\Delta$ only in the first row and similarly $\tilde{\mathbf{I}}$ differ from $\mathbf{I}$ only in the first row. Therefore

$$
\text{rank}(\tilde{\Delta}^T \mathbf{M} - \tilde{\mathbf{I}} \mathbf{M} \Delta) \leq \text{rank}(\Delta^T \mathbf{M} - \mathbf{M} \Delta) + 1.
$$

Then since the Sylvester-type displacement rank of $\mathbf{M}$ with respect to $(\Delta^T, \Delta)$ is at most $r$, we have

$$
\text{rank}(\mathbf{M} \mathbf{J} - \Delta \mathbf{M} \mathbf{J}^T) \leq r + 1.
$$

In a similar fashion, one can verify that

$$
\text{rank}(\mathbf{J} \mathbf{M} - \Delta \mathbf{J} \mathbf{M}^T) \leq r + 1.
$$
Algorithm 15: Symmetric Factorization of General Matrices with Small Sylvester-Type Displacement Rank

1. **Input:** Hankel matrix $M \in \mathbb{F}^{2k \times 2k}$
2. Set $M_0 = M$
3. for $t = 1, \ldots, \frac{k}{2}$ do
   4. Set $N_t = i \cdot \text{imag}(\bar{S}_t M_{t-1} \bar{S}_t^*)$
   5. Set $M_t = \text{real}(\bar{S}_t M_{t-1} \bar{S}_t^*)$
   6. Set $N_{\frac{k}{2}+1} = M_{\frac{k}{2}}$
7. for $t = 1, \ldots, \frac{k}{2} + 1$ do
   8. Set $X_t$ and $Y_t$ to be matrices such that $N_t = X_t X_t^* - Y_t Y_t^*$
   9. Set $B_t = \bar{S}_1 \bar{S}_2^* \cdots \bar{S}_{t-1} \bar{S}_t^* X_t$ and $C_t = \bar{S}_1 \bar{S}_2^* \cdots \bar{S}_{t-1} \bar{S}_t^* Y_t$
10. return $[B_{\frac{k}{2}+1}, B_{\frac{k}{2}} \cdots B_2 \ B_1]$ and $[C_{\frac{k}{2}+1}, C_{\frac{k}{2}} \cdots C_2 \ C_1]$}

Therefore

$$\text{rank}((MJ - JM) - \Delta (MJ - JM) \Delta^T) \leq \text{rank}(MJ - \Delta MJ \Delta^T) + \text{rank}(JM - \Delta JM \Delta^T) \leq 2r + 2.$$  

Finally note that since $M$ is symmetric $M_{i,n+1-i} = M_{n+1-i,i}$ for all $i \in [n]$. Therefore since $(MJ)_{i,i} = M_{i,n+1-i}$ and $(JM)_{i,i} = M_{n+1-i,i}$, we have $(MJ - JM)_{i,i} = 0$. Thus the diagonal entries of $\text{imag}(SMSS^*)$ are zero.

We are now equipped to prove our result for general matrices with small displacement ranks. We state the theorem for the inverse of Hankel matrices, but our algorithm and proof work for these general matrices due to Lemma 3.7.0.1.

**Theorem 3.2.0.5.** Let $H \in \mathbb{R}^{n \times n}$ be a Hankel matrix with bit complexity $\ell$ and condition number bounded by $2^{\ell}$. There exists an algorithm that finds a representation of matrices $B$ and $C$, each with $n$ rows, $O(n \log n)$ columns, and bit complexity $\ell$ in time $\tilde{O}(n^{\omega/2} \cdot \ell)$ such that $\|H^{-1} - (BB^* - CC^*)\|_F < \frac{1}{2^\ell}$.

**Proof.** We consider a general matrix $M$ that has Sylvester-type displacement rank of two with respect to $(\Delta, \Delta^T)$ and $(\Delta^T, \Delta)$. For example, one can consider $M = H^{-1}$. Note that
we can find a representation of $H^{-1}$ as $XY^\top$ in $\tilde{O}(n \cdot \ell)$ time using the approach of [45].

Without loss of generality, we assume that $n = 2^k$ and $k$ is even because otherwise, we can extend the matrix to a size of power of four by appropriately copying the entries to make sure Sylvester-type displacement rank does not change.

We show that Algorithm 15 outputs the desired factorization in the specified running time and bit complexity. Note that this algorithm is similar to the one we used for the symmetric factorization of Hankel matrices (Algorithm 14). The main difference is that we recurse only for $\frac{k}{2}$ iterations. This is because by Lemma 3.7.0.1, the blocks of $M_t$ have a Sylvester-type displacement rank of $2^{t+1} + 1$ and a size of $\frac{n}{2} \times \frac{n}{2}$. So for $t = \frac{k}{2}$, the size of each block is $\sqrt{n} \times \sqrt{n}$ while its displacement rank is $2\sqrt{n}$. In other words, the displacement rank is more than the rank of the matrix and therefore it does not help with speeding up the computation of symmetric factorization. The rest of the proof basically is similar to the symmetric factorization of Hankel matrices.

Similar to Lemmas 3.6.0.2 and 3.6.0.3, $N_t$ consists of $2^{t-1} \times 2^{t-1}$ Hermitian blocks. However instead of being Toeplitz, by Lemma 3.7.0.1 each block has a Stein-type displacement rank of $2^{t+1} + 2$ with respect to $(\Delta, \Delta^\top)$. Moreover each block is persymmetric. The structure of the blocks is also similar to Lemma 3.6.0.3. More specifically, the first block row of $N_t$ consists of $2^{t-1}$ matrices

$$\begin{bmatrix} Q_1 & Q_2 & \cdots & Q_{2^{t-1}} & Q_{2^{t-1}} \end{bmatrix}.$$ 

The second block-row is

$$\begin{bmatrix} Q_2 & JQ_1J & Q_4 & JQ_3J & \cdots & Q_{2^{t-1}} & JQ_{2^{t-1}-1}J \end{bmatrix}$$

For $s = 1, \ldots, t - 2$, let $P_s$ be the matrix consisting of block-rows 1 to $2^s$, and $\bar{P}_s$ be the
matrix consisting of block-rows $2^s + 1$ to $2^{s+1}$. Let

$$P_s = \begin{bmatrix} P_{s,1} & P_{s,2} & \cdots & P_{s,2^{t-s}} \end{bmatrix}. $$

Then

$$\tilde{P}_s = \begin{bmatrix} P_{s,2} & P_{s,1} & P_{s,4} & P_{s,3} & \cdots & P_{s,2^{t-s}} & P_{s,2^{t-s}-1} \end{bmatrix}. $$

The structure of block columns of $N_t$ is also similar to the above. For example, the matrix consisting only of the first row and the first column of $N_t$ is like the following.

$$Z_t = \begin{bmatrix} Q_1 & Q_2 & Q_3 & \cdots & Q_{2^{t-1}} \\ Q_2 & 0 & 0 & \cdots & 0 \\ Q_3 & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ Q_{2^{t-1}} & 0 & 0 & \cdots & 0 \end{bmatrix}. $$

All the other block-rows and block-columns are obtained by taking permutations of the first block-row and the first block-column, respectively. Moreover the block-row and the block-column with the same index are obtained by the same permutation. Since $Q_1$ is a Hermitian matrix, $Q_1 - \Delta Q_1 \Delta^T$ is also Hermitian. Therefore the eigenvalues of $Q_1 - \Delta Q_1 \Delta^T$ are real and since it is an $\frac{n}{2} \times \frac{n}{2}$ of rank $2^{t+1} + 2$, we can find its eigenvalue decomposition in $O(\frac{n}{2} \cdot 2^{t-\omega-1} \cdot t)$. Then we can separate the eigenvectors associated with positive and negative eigenvalues into matrices $\tilde{D}$ and $\tilde{E}$, respectively, and write $Q_1 - \Delta Q_1 \Delta^T$ as $\tilde{D}\tilde{D}^* - \tilde{E}\tilde{E}^*$. Then by writing

$$Q_1 = Q_1 - \Delta Q_1 \Delta^T + \Delta(Q_1 - \Delta Q_1 \Delta^T)\Delta^T + \Delta \Delta(Q_1 - \Delta Q_1 \Delta^T)\Delta^T \Delta^T + \cdots ,$$
we can take the appropriate shifts and write $Q = DD' - CC^*$, where

$$D = \begin{bmatrix} D & \Delta D & \Delta \Delta D & \cdots \end{bmatrix}, \quad \text{and} \quad E = \begin{bmatrix} E & \Delta E & \Delta \Delta E & \cdots \end{bmatrix}.$$

Then we can write $Z_t$ as $F_1 F_1^* - G_1 G_1^*$, where

$$F_1 = \begin{bmatrix} D & I & 0 \\ 0 & Q_2 & 0 \\ 0 & Q_3 & 0 \\ \vdots & \vdots & \vdots \\ 0 & Q_{2t-1} & 0 \end{bmatrix}, \quad \text{and} \quad G_1 = \begin{bmatrix} E & I & 0 \\ 0 & 0 & Q_2 \\ 0 & 0 & Q_3 \\ \vdots & \vdots & \vdots \\ 0 & 0 & Q_{2t-1} \end{bmatrix}.$$

Then since the rest of block-rows and block-columns of $N_t$ are obtained by permuting its first block-row and block-column, we can obtain the factorization by taking appropriate shift and permutations of $F_1$ and $G_1$. This then produces $\bar{B}_t$ and $\bar{C}_t$ such that $N_t = \bar{B}_t \bar{B}_t^* - \bar{C}_t \bar{C}_t^*$. Then by defining $B_t = \bar{S}_1^* \bar{S}_2^* \cdots \bar{S}_{t-1}^* \bar{S}_t^* \bar{B}_t$ and $C_t = \bar{S}_1^* \bar{S}_2^* \cdots \bar{S}_{t-1}^* \bar{S}_t^* \bar{C}_t$, and $B = \begin{bmatrix} B_{\frac{t}{2}+1} & B_{\frac{t}{2}} & \cdots & B_2 & B_1 \end{bmatrix}$ and $C = \begin{bmatrix} C_{\frac{t}{2}+1} & C_{\frac{t}{2}} & \cdots & C_2 & C_1 \end{bmatrix}$, we have

$$M = BB^* - CC^*.$$

Note that for each $N_t$, we only need to compute an eigendecomposition for its corresponding $Q_1 - \Delta Q_1 \Delta^\top$, and the rest of the decomposition for $N_t$ follows from deterministic shifts and permutations. We do not even need to compute $Q_2, \cdots, Q_{2t-1}$ since the entries of them can be obtained by $O(\log n)$ addition/subtraction of the entries of the original matrix. Since $Q_1$ is obtained by taking summations over principal submatrices of the original matrix, its bit complexity and operator norm are the same as the original matrix (up to a $\log n$ factor). Therefore the cost of computing this eigendecomposition is

$$\tilde{O}(\frac{n}{2^t} \cdot 2^{t(\omega-1)} \cdot t).$$
because $Q_1 - \Delta Q_1 \Delta^\top$ is an $\frac{n^2}{2^t} \times \frac{n^2}{2^t}$ of rank $2^{t+1} + 2$, and we can add a random matrix with small entries (of less than $\frac{1}{n^2 \ell}$). Adding this random matrix, does not cause us to go above the error threshold but it causes a gap between the eigenvalues of $Q_1 - \Delta Q_1 \Delta^\top$ and causes it to have $\text{poly} \ n \cdot 2^t$ condition number which results in the running time stated above — see [128]. Therefore the total cost of computing the representation is

$$1 + \frac{k}{2} \sum_{t=1}^{1+\frac{k}{2}} \tilde{O}(\frac{n}{2^t} \cdot 2^t (\omega^{-1}) \cdot \ell) = \sum_{t=1}^{1+\frac{k}{2}} \tilde{O}(n \cdot 2^t (\omega^{-2}) \cdot \ell).$$

Since $t \leq \frac{k}{2} + 1$ and $n^{1+\frac{k}{2}} = 2 \sqrt{n}$, this is bounded by $\tilde{O}(n^{\omega/2} \cdot \ell).$  

3.8 Discussion and Conclusion

In this chapter, we presented novel super-fast algorithms to find a representation of symmetric factorizations of the form $BB^* - CC^*$ for Hankel matrices and their inverses. Our running times for Hankel matrices and their inverses are $\tilde{O}(n \cdot \ell)$ and $\tilde{O}(n^{\omega/2} \cdot \ell)$. We also conjectured that it is possible to find factorizations of the form $BB^*$ for these problems in the same running times. We explained how our conjectures lead to faster algorithms for solving a batch of linear systems faster than the approach of [45] and how they lead to a faster-than-matrix-multiplication algorithm for solving sparse poly-conditioned linear programs.

Here we present a statement that has the same implications. This is weaker than our conjectures but stronger than the results we proved. Suppose we find $B$ and $C$ such that $BB^* - CC^* = (K^\top AK)^{-1}$ and $BB^* \preceq \text{poly}(n)(K^\top AK)^{-1}$, then since $BB^* - CC^* = (K^\top AK)^{-1} \succeq 0$, $CC^* \preceq \text{poly}(n)(K^\top AK)^{-1}$. Then by the argument of (Equation 3.3.5), $KB$ and $KC$, both will have low bit-complexity since

$$KBB^*K^\top \preceq \text{poly}(n)K(K^\top AK)^{-1}K^\top = \text{poly}(n)A^{-1}.$$
and

\[ KCC^* K^T \preceq \text{poly}(n) K(K^T AK)^{-1} K^T = \text{poly}(n) A^{-1}. \]

Then using the linear operator \( \bar{B}\bar{B}^* - \bar{C}\bar{C}^* \) for \( \bar{B} = KB \) and \( \bar{C} = KC \) leads to the same running times presented in (Equation 3.3.6) and (Equation 3.3.9).
4.1 Introduction

Tensor decomposition has a rich multidisciplinary history with countless applications in data mining, machine learning, and signal processing [129, 130, 131, 132]. The most widely-used tensor decompositions are the CP decomposition and the Tucker decomposition. Similar to the singular value decomposition of a matrix, both decompositions have natural analogs of low-rank structure. Unlike matrix factorization, however, computing the rank of a tensor and the best rank-one tensor are NP-hard [60]. Therefore, most low-rank tensor decomposition algorithms decide on the rank structure in advance, and then optimize the variables of the decomposition to fit the data. While conceptually simple, this approach
is extremely effective in practice for many applications.

The alternating least squares (ALS) algorithm is the main workhorse for low-rank tensor decomposition, e.g., it is the first algorithm mentioned in the MATLAB Tensor Toolbox [133]. For both CP and Tucker decompositions, ALS cyclically optimizes disjoint blocks of variables while keeping all others fixed. As the name suggests, each step solves a linear regression problem. The core tensor update step in ALS for Tucker decompositions is notoriously expensive but highly structured. In fact, the design matrix of this regression problem is the Kronecker product of the factor matrices of the Tucker decomposition $\mathbf{K} = \mathbf{A}^{(1)} \otimes \cdots \otimes \mathbf{A}^{(N)}$. Our work builds on a line of Kronecker regression algorithms [134, 56, 135] to give the first subquadratic-time algorithm for solving Kronecker regression to a $(1 + \varepsilon)$-approximation while avoiding an exponential term of $O(\varepsilon^{-N})$ in the running time.

We combine leverage score sampling, iterative methods, and a novel way of multiplying sparsified Kronecker product matrices to fully exploit the Kronecker structure of the design matrix. We also extend our approach to block-design matrices where one block is a Kronecker product, achieving subquadratic-time algorithms for (1) Kronecker ridge regression and (2) updating the factor matrix of a Tucker decomposition in ALS, which is not a pure Kronecker regression problem. Putting everything together, this chapter improves the running time of all steps of ALS for Tucker decompositions and runs in time that is sublinear in the size of the input tensor, linear in the error parameter $\varepsilon^{-1}$, and subquadratic in the number of columns of the design matrix in each step. Our algorithms support L2 regularization in the Tucker loss function, so the decompositions can readily be used in downstream learning tasks, e.g., using the factor matrix rows as embeddings for clustering [136]. Regularization also plays a critical role in the more general tensor completion problem to prevent overfitting when data is missing and has applications in differential privacy [137, 138].

The current-fastest Kronecker regression algorithm of [56] uses leverage score sampling and achieves the following running times for $\mathbf{A}^{(n)} \in \mathbb{R}^{I_n \times R_n}$ with $I_n \geq R_n$, for all
$n \in [N]$, where $R = \prod_{n=1}^{N} R_n$ and $\omega < 2.373$ denotes the matrix multiplication exponent [16]:

1. $\tilde{O}(\sum_{n=1}^{N} (\text{nnz}(A^{(n)}) + R_n^\omega + R^\omega \epsilon^{-1})$ by sampling $\tilde{O}(R \epsilon^{-1})$ rows of $K$ by their leverage scores.

2. $\tilde{O}(\sum_{n=1}^{N} (\text{nnz}(A^{(n)}) + R_n^\omega \epsilon^{-1} + Re^{-N})$ by sampling $\tilde{O}(R_n \epsilon^{-1})$ rows from each factor matrix $A^{(n)}$ and taking the Kronecker product of the sampled factor matrices.

Note that the second approach is linear in $R$, but the error parameter has an exponential cost in the number of factor matrices. In this chapter, we show that the running time of the first approach can be improved to subquadratic in $R$ without increasing the running time dependence on $\epsilon$ in the dominant term, simultaneously improving on both approaches.

**Theorem 4.1.0.1.** For $n \in [N]$, let $A^{(n)} \in \mathbb{R}^{I_n \times R_n}$, $I_n \geq R_n$, and $b \in \mathbb{R}^{I_1 \cdots I_n}$. There is a $(1 + \epsilon)$-approximation algorithm for solving $\arg \min_x \| (A^{(1)} \otimes \cdots \otimes A^{(N)})x - b \|_2^2$ that runs in time

$$\tilde{O} \left( \sum_{n=1}^{N} (\text{nnz}(A^{(n)}) + R_n^\omega N^2 \epsilon^{-2}) + \min_{S \subseteq [N]} \text{MM} \left( \prod_{n \in S} R_n, R \epsilon^{-1}, \prod_{n \in [N] \setminus S} R_n \right) \right),$$

where $\text{MM}(a, b, c)$ is the running time of multiplying an $a \times b$ matrix with a $b \times c$ matrix.

If we do not use fast matrix multiplication ([104, 16]), the last term in (Equation 4.1.1) is $\tilde{O}(R^2 \epsilon^{-1})$, which is already an improvement over the standard $\tilde{O}(R^3 \epsilon^{-1})$ running time.

With fast matrix multiplication, $\text{MM}(\prod_{n \in S} R_n, R \epsilon^{-1}, \prod_{n \in [N] \setminus S} R_n)$ is subquadratic in $R$ for any nontrivial subset $S \not\in \{\emptyset, [N]\}$, which is an improvement over $\tilde{O}(R^\omega \epsilon^{-1}) \approx \tilde{O}(R^{2.373} \epsilon^{-1})$. If there exists a “balanced” subset $S$ such that $\prod_{n \in S} R_n \approx \sqrt{R}$, our running time goes as low as $\tilde{O}(R^{1.626} \epsilon^{-1})$ using [104]. For ease of notation, we denote the subquadratic improvement by the constant $\theta^* > 0$, where

$$R^{2-\theta^*} = \min_{S \subseteq [N]} \text{MM}(\prod_{n \in S} R_n, R, \prod_{n \in [N] \setminus S} R_n).$$
Updating the core tensor in the ALS algorithm for Tucker decomposition is a pure Kronecker product regression as described in Theorem 4.1.0.1, but updating the factor matrices is a regression problem of the form \( \arg \min_x \| K M x - b \|_2^2 \), where \( K \) is a Kronecker product and \( M \) is a matrix without any particular structure. We show that such problems can be converted to block regression problems where one of the blocks is \( K \). We then develop sublinear-time leverage score sampling techniques for these block matrices, which leads to the following theorem that accelerates all of the ALS steps.

**Theorem 4.1.0.2.** There is an ALS algorithm for L2-regularized Tucker decompositions that takes a tensor \( \mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_N} \) and returns \( N \) factor matrices \( A^{(n)} \in \mathbb{R}^{I_n \times R_n} \) and a core tensor \( G \in \mathbb{R}^{R_1 \times \cdots \times R_n} \) such that each factor matrix and core update is a \((1 + \varepsilon)\)-approximation to the optimum with high probability. The running times of each step are:

- **Factor matrix** \( A^{(k)} \): \( \tilde{O}(\sum_{n=1}^{N} (\text{nnz}(A^{(n)}) + R_n^\omega N^2 \varepsilon^{-2}) + I_k R^{2-\theta^*} \varepsilon^{-1} + I_k R \sum_{n=1}^{N} R_n + R_k^\omega) \),

- **Core tensor** \( G \): \( \tilde{O}(\sum_{n=1}^{N} (\text{nnz}(A^{(n)}) + R_n^\omega N^2 \varepsilon^{-2}) + R^{2-\theta^*} \varepsilon^{-1}) \),

where \( R = \prod_{n=1}^{N} R_n \), \( R_{\neq k} = R / R_k \), and \( \theta^* > 0 \) is a constant derived from fast rectangular matrix multiplication.

For tensors of even modest order, the superlinear term in \( R \) is the bottleneck in many applications since \( R \) is exponential in the order of the tensor. It follows that our improvements are significant in both theory and practice as illustrated in our experiments in Section 4.6.

### 4.1.1 Our Contributions and Techniques

We present several new results about approximate Kronecker regression and the ALS algorithm for Tucker decompositions. Below is a summary of our contributions:

1. Our main technical contribution is the algorithm \texttt{FastKroneckerRegression} in Section 4.4. This Kronecker regression algorithm builds on the block-sketching tools introduced in Section 4.3, and combines iterative methods with a fast novel
Table 4.1: Running times of TuckerALS (Algorithm 16) factor matrix and core tensor updates for \( \lambda = 0 \) using different Kronecker regression methods. The factor matrices are denoted by \( A^{(n)} \in \mathbb{R}^{I_n \times R_n} \). The input tensor has size \( I = I_1 \cdots I_N \) and the core tensor has size \( R = R_1 \cdots R_N \). Let \( I_{\neq k} = I/I_k \) and \( R_{\neq k} = R/R_k \). We use \( \omega < 2.373 \) for the matrix-multiplication exponent and the constant \( \theta^* > 0 \) for the optimally balanced fast rectangular matrix multiplication as stated in Theorem 4.4.2.4, i.e., \( R^2 - \theta^* = \min_{T \subseteq [N]} \text{MM} \left( \prod_{n \in T} R_n, R, \prod_{n \notin T} R_n \right) \). Factors of \( N \) are dropped for notational brevity.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Factor matrix ( A^{(k)} )</th>
<th>Core tensor ( G )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Naive</td>
<td>( O \left( I_k R R_{\neq k} + I_k R R_k + R_k^{\omega} + I R_{\neq k} \right) )</td>
<td>( O \left( R^{\omega} + I R \right) )</td>
</tr>
<tr>
<td>This chapter (Lemma 4.4.2.1)</td>
<td>( O \left( I_k R (\sum_{n=1}^{N} R_n) + R_k^{\omega} + I (\sum_{n \neq k} R_n) + I_k R_k^2 \right) )</td>
<td>( O \left( R^2 + I \sum_{n=1}^{N} R_n \right) )</td>
</tr>
<tr>
<td>This chapter (Theorem 4.1.0.2)</td>
<td>( \tilde{O} \left( I_k R^{2-\theta^*} e^{-1} + I_k R (\sum_{n=1}^{N} R_n) + R_k^{\omega} e^{-2} \right) )</td>
<td>( \tilde{O} \left( R^{2-\theta^*} e^{-1} \right) )</td>
</tr>
<tr>
<td>[56]</td>
<td>---</td>
<td>( \tilde{O} \left( R^{\omega} e^{-2} \right) )</td>
</tr>
</tbody>
</table>

Kronecker-matrix multiplication for sparse vectors and matrices and fast rectangular matrix multiplication to achieve a running time that is subquadratic in the number of columns in the Kronecker matrix. A key insight is to use the original (non-sketch) Kronecker product as the preconditioner in the Richardson iterations when solving the sketched problem. This, by itself, improves the running time to quadratic. Then to achieve subquadratic running time, we exploit the singular value decomposition of Kronecker products and present a novel method for multiplying a sparsified Kronecker product matrix (Lemma 4.4.2.1 and Theorem 4.4.2.4).

2. We generalize our Kronecker regression techniques to work for Kronecker ridge regression and the factor matrix updates in ALS for Tucker decomposition. We show that a factor matrix update is equivalent to solving an equality-constrained Kronecker regression problem with a low-rank update to the preconditioner in the Richardson iterations. We can implement these new matrix-vector products nearly as fast by using the Woodbury matrix identity. Thus, we provably speed up each step of Tucker ALS, i.e., the core tensor and factor matrices.

3. We give a block-sketching toolkit in Section 4.3 that states we can sketch blocks of a matrix by their leverage scores, i.e., their leverage scores in isolation, not with respect
to the entire block matrix. This is one of the ways we exploit the Kronecker product structure of the design matrix. This approach can be useful for constructing spectral approximations and for approximately solving block regression problems. One corollary is that we can use the “sketch-and-solve” method for any ridge regression problem (Corollary 4.3.0.5).

4. We compare \texttt{FastKroneckerRegression} with [56, Algorithm 1] on a synthetic Kronecker regression task studied in [134, 56] and as a subroutine in ALS for computing the Tucker decomposition of image tensors [139, 58, 140]. Our results show the importance of reducing the running time dependence on the number of columns in the Kronecker product.

4.1.2 Related Work

\textbf{Kronecker Regression.} [134] recently gave the first Kronecker regression algorithm based on \texttt{TensorSketch} [141] that is faster than forming the Kronecker product. [56] improved this by removing the dependence on \(O(\text{nnz}(b))\) from the running time, where \(b \in \mathbb{R}^{I_1 \cdots I_N}\) is the response vector. [142] recently initiated the study of dynamic Kronecker regression, where the factor matrices \(A^{(n)}\) undergo updates and the solution vector can be efficiently queried. [135] studied the generalized Kronecker regression problem. Very recently, [37] analyzed the bit complexity of iterative methods with preconditioning for solving linear regression problems under fixed-point arithmetic. They show that the actual running time of such algorithms (i.e., the number of bit operations) is at most a factor of \(\log(\kappa) \cdot \log(1/\epsilon)\) more than the number of arithmetic operations, where \(\kappa\) is the condition number of the design matrix and \(\epsilon\) is the error parameter. This result applies to our work, too; however, for the rest of this exposition, we discuss the number of arithmetic operations. Finally, note that \(\kappa(A \otimes B) = \kappa(A) \cdot \kappa(B)\).
Ridge Leverage Scores. [143] extended the notion of statistical leverage scores to account for L2 regularization. Sampling from approximate ridge leverage score distributions has since played a key role in sparse low-rank matrix approximation [144], the Nyström method [145], bounding statistical risk in ridge regression [146], and ridge regression [147, 146, 148, 149]. Fast recursive algorithms for computing approximate leverage scores [40] and for solving overconstrained least squares [150] are also closely related.

Tensor Decomposition. [151] and [152] used leverage score sampling to speed up ALS for CP decomposition.1 [153] gave a polynomial-time, relative-error approximation algorithm for several low-rank tensor decompositions, which include CP and Tucker. [154] showed that if the tensor has an exact Tucker decomposition, then all local minima are globally optimal. Randomized low-rank Tucker decompositions based on sketching have become increasingly popular, especially in streaming applications: [155, 156, 157, 158, 132, 159, 139, 160]. The more general problem of low-rank tensor completion is also a fundamental approach for estimating the values of missing data [161, 162, 163, 164, 165]. Fundamental algorithms for tensor completion are based on ALS [166, 167, 168], Riemannian optimization [169, 170, 171], or projected gradient methods [172]. Optimizing the core shape of a Tucker decomposition subject to a memory constraint or reconstruction error guarantees has also been studied recently [62, 61, 173, 174].

4.2 Preliminaries

Notation. The order of a tensor is the number of its dimensions. We denote scalars by normal lowercase letters $x \in \mathbb{R}$, vectors by boldface lowercase letters $\mathbf{x} \in \mathbb{R}^n$, matrices by boldface uppercase letters $X \in \mathbb{R}^{m \times n}$, and higher-order tensors by boldface script letters $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \ldots \times I_N}$. We use normal uppercase letters to denote the size of an index set (e.g., $I$).

---

1The design matrix in each step of ALS for CP decomposition is a Khatri–Rao product, not a Kronecker product. CP decomposition does not suffer from a bottleneck step like ALS for Tucker decomposition since it is a sparser decomposition, i.e., CP decomposition does not have a core tensor—just factor matrices.
Algorithm 16: TuckerALS

1. **Input:** $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$, core shape $(R_1, R_2, \ldots, R_N, \lambda$)
2. Initialize core tensor $G \in \mathbb{R}^{R_1 \times R_2 \times \cdots \times R_N}$
3. Initialize factors $A^{(n)} \in \mathbb{R}^{I_n \times R_n}$ for $n \in [N$
4. while has not converged do
5. for $n = 1$ to $N$ do
6. $K \leftarrow A^{(1)} \otimes \cdots \otimes A^{(n-1)} \otimes A^{(n+1)} \otimes \cdots \otimes A^{(N)}$
7. $B \leftarrow X^{(n)}$
8. for $i = 1$ to $I_n$ do
9. $y^* \leftarrow \arg\min_y \|KG^{(n)}_i y - b_i^\top\|_2^2 + \lambda \|y\|_2^2$
10. Update factor row $a_i^{(n)} \leftarrow y^\top$
11. $K \leftarrow A^{(1)} \otimes A^{(2)} \otimes \cdots \otimes A^{(N)}$
12. $g^* \leftarrow \arg\min_g \|Kg - \text{vec}(\mathcal{X})\|_2^2 + \lambda \|g\|_2^2$
13. Update core tensor $G \leftarrow \text{vec}^{-1}(g^*)$
14. return $G, A^{(1)}, A^{(2)}, \ldots, A^{(N)}$

$[N] = \{1, 2, \ldots, N\}$). The $i$-th entry of a vector $x$ is denoted by $x_i$, the $(i, j)$-th entry of a matrix $X$ by $x_{ij}$, and the $(i, j, k)$-th entry of a third-order tensor $\mathcal{X}$ by $x_{ijk}$.

**Linear Algebra.** Let $I_n$ denote the $n \times n$ identity matrix and $0_{m \times n}$ denote the $m \times n$ zero matrix. The transpose of $A \in \mathbb{R}^{m \times n}$ is $A^\top$, the Moore–Penrose inverse (also called pseudoinverse) is $A^+$, and the spectral norm is $\|A\|_2$. The singular value decomposition (SVD) of $A$ is a factorization of the form $U\Sigma V^\top$, where $U \in \mathbb{R}^{m \times m}$ and $V \in \mathbb{R}^{n \times n}$ are orthogonal matrices, and $\Sigma \in \mathbb{R}^{m \times n}$ is a non-negative real diagonal matrix. The entries $\sigma_i(A)$ of $\Sigma$ are the singular values of $A$, and the number of non-zero singular values is equal to $r = \text{rank}(A)$. The compact SVD is a related decomposition where $\Sigma \in \mathbb{R}^{r \times r}$ is a diagonal matrix containing the non-zero singular values. The Kronecker product of two matrices $A \in \mathbb{R}^{m \times n}$ and $B \in \mathbb{R}^{p \times q}$ is denoted by $A \otimes B \in \mathbb{R}^{(mp) \times (nq)}$.

**Tensor Products.** Fibers of a tensor are the vectors we get by fixing all but one index. If $\mathcal{X}$ is a third-order tensor, we denote the column, row, and tube fibers by $x_{.:jk}$, $x_{i:.}$, and $x_{ij:}$, respectively. The mode-$n$ unfolding of a tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ is the matrix
$X_{(n)} \in \mathbb{R}^{I_n \times (I_1 \cdot I_{n-1} I_{n+1} \cdots I_N)}$ that arranges the mode-$n$ fibers of $\mathcal{X}$ as columns of $X_{(n)}$ ordered lexicographically by index. The vectorization of $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ is the vector $\text{vec}(\mathcal{X}) \in \mathbb{R}^{I_1 I_2 \cdots I_N}$ formed by vertically stacking the entries of $\mathcal{X}$ ordered lexicographically by index. For example, this transforms $X \in \mathbb{R}^{m \times n}$ into a tall vector $\text{vec}(X)$ by stacking its columns. We use $\text{vec}^{-1}(x)$ to undo this operation when it is clear from context what the shape of the output tensor should be.

The $n$-mode product of tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ and matrix $A \in \mathbb{R}^{J \times I_n}$ is denoted by $\mathcal{Y} = \mathcal{X} \times_n A$ where $\mathcal{Y} \in \mathbb{R}^{I_1 \times \cdots I_n-1 \times J \times I_{n+1} \times \cdots \times I_N}$. This operation multiplies each mode-$n$ fiber of $\mathcal{X}$ by the matrix $A$. This operation is expressed elementwise as

$$(\mathcal{X} \times_n A)_{i_1 \ldots i_{n-1} j_{n+1} \ldots i_N} = \sum_{i_n=1}^{I_n} x_{i_1 i_2 \ldots i_N} a_{j_{n}}.$$ 

The Frobenius norm $\|\mathcal{X}\|_F$ of a tensor $\mathcal{X}$ is the square root of the sum of the squares of its entries.

**Tucker Decomposition.** The Tucker decomposition decomposes tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ into a core tensor $G \in \mathbb{R}^{R_1 \times R_2 \times \cdots \times R_N}$ and $N$ factor matrices $A^{(n)} \in \mathbb{R}^{I_n \times R_n}$. Given a regularization parameter $\lambda \in \mathbb{R}_{\geq 0}$, we compute a Tucker decomposition by minimizing the nonconvex loss function

$$L\left(G, A^{(1)}, \ldots, A^{(N)}; \mathcal{X}\right) = \left\| \mathcal{X} - G \times_1 A^{(1)} \cdots \times_N A^{(N)} \right\|_F^2 + \lambda \left( \left\| G \right\|_F^2 + \sum_{n=1}^{N} \left\| A^{(n)} \right\|_F^2 \right).$$

Entries of the reconstructed tensor $\widehat{\mathcal{X}} \overset{\text{def}}{=} G \times_1 A^{(1)} \times_2 \cdots \times_N A^{(N)}$ are

$$\widehat{x}_{i_1 i_2 \ldots i_N} = \sum_{r_1=1}^{R_1} \cdots \sum_{r_N=1}^{R_N} g_{r_1 r_2 \ldots r_N} a^{(1)}_{i_1 r_1} \cdots a^{(N)}_{i_N r_N}. \quad (4.2.1)$$

(Equation 4.2.1) demonstrates that $\widehat{\mathcal{X}}$ is the sum of $R_1 \cdots R_N$ rank-1 tensors. The tuple $(R_1, R_2, \ldots, R_N)$ is the multilinear rank of the decomposition. The multilinear rank is
typically chosen in advance and much smaller than the dimensions of $\mathcal{X}$.

**Alternating Least Squares.** We present TuckerALS in Algorithm 16 and highlight its connections to Kronecker regression. The core tensor update (Lines 10–12) is a ridge regression problem where the design matrix $K_{\text{core}} \in \mathbb{R}^{I_1 \cdots I_N \times R_1 \cdots R_N}$ is a Kronecker product of the factor matrices. Each factor matrix update (Lines 5–9) also has Kronecker product structure, but there are additional subspace constraints we must account for. We describe these constraints in more detail in Section 4.5.

### 4.3 Row Sampling and Approximate Regression

Here we establish our sketching toolkit. The $\lambda$-ridge leverage score of the $i$-th row of $A \in \mathbb{R}^{n \times d}$ is

$$\ell_{i}^\lambda(A) \overset{\text{def}}{=} a_i : (A^\top A + \lambda I)^+ a_i^\top. \quad (4.3.1)$$

The matrix of cross $\lambda$-ridge leverage scores is $A(A^\top A + \lambda I)^+ A^\top$. We denote its diagonal by $\ell^\lambda(A)$ because it contains the $\lambda$-ridge leverage scores of $A$. Ridge leverage scores generalize statistical leverage scores in that setting $\lambda = 0$ gives the leverage scores of $A$. We denote the vector of statistical leverage scores by $\ell(A)$. If $A = U \Sigma V^\top$ is the compact SVD of $A$, then for all $i \in [n]$, we have

$$\ell_{i}^\lambda(A) = \sum_{k=1}^r \frac{\sigma_k^2(A)}{\sigma_k^2(A) + \lambda} u_{ik}^2, \quad (4.3.2)$$

where $r = \text{rank}(A)$. It follows that every $\ell_{i}^\lambda(A) \leq 1$ since $U$ is an orthogonal matrix. We direct the reader to [143] or [40] for further details.

The main results in this chapter build on approximate leverage score sampling for block matrices. The $\lambda$-ridge leverage scores of $A \in \mathbb{R}^{n \times d}$ can be computed by appending $\sqrt{\lambda} I_d$ to the bottom of $A$ to get $\overline{A} \in \mathbb{R}^{(n+d) \times d}$ and considering the leverage scores of $\overline{A}$, so we state the following results in terms of statistical leverage scores without loss of generality.
**Definition 4.3.0.1.** For any $A \in \mathbb{R}^{n \times d}$, the vector $\hat{\ell}(A) \in \mathbb{R}^n$ is a $\beta$-overestimate for the leverage score distribution of $A$ if, for all $i \in [n]$, it satisfies

$$\frac{\hat{\ell}_i(A)}{\|\hat{\ell}(A)\|_1} \geq \beta \frac{\ell_i(A)}{\|\ell(A)\|_1} = \beta \frac{\ell_i(A)}{\text{rank}(A)}.$$ 

Next we describe the approximate leverage score sampling algorithm in [175, Section 2.4]. The core idea here is that if we sample $\tilde{O}(d/\beta)$ rows and reweight them appropriately, this smaller sketched matrix can be used instead of $A$ to give provable guarantees for many problems.

**Definition 4.3.0.2 (Leverage score sampling).** Let $A \in \mathbb{R}^{n \times d}$ and $p \in [0, 1]^n$ be a $\beta$-overestimate for the leverage score distribution of $A$ such that $\|p\|_1 = 1$. We denote the following procedure with $\text{SampleRows}(A, s, p)$. Initialize sketch matrix $S = 0_{s \times n}$. For each row $i$ of $S$, independently and with replacement, select an index $j \in [n]$ with probability $p_j$ and set $s_{ij} = 1/\sqrt{p_j s}$. Return sketch $S$.

The main result in this section is that we can choose to sketch a single block of a matrix by the leverage scores of that block in isolation. This sketched submatrix can then be used with the other (non-sketched) block to give a spectral approximation to the original matrix or for approximate linear regression. The notation $A \preceq B$ is the Loewner order and means $B - A$ is positive semidefinite.

**Lemma 4.3.0.3.** Let $A = \begin{bmatrix} A_1; A_2 \end{bmatrix}$ be vertically stacked with $A_1 \in \mathbb{R}^{n_1 \times d}$ and $A_2 \in \mathbb{R}^{n_2 \times d}$. Let $p \in [0, 1]^{n_1}$ be a $\beta$-overestimate for the leverage score distribution of $A_1$. If $s > 144d \ln(2d/\delta)/(\beta\varepsilon^2)$, the sketch $S$ returned by $\text{SampleRows}(A_1, s, p)$ guarantees, with probability at least $1 - \delta$, that

$$(1 - \varepsilon)A^T A \preceq (SA_1)^T SA_1 + A_2^T A_2 \preceq (1 + \varepsilon)A^T A.$$
Lemma 4.3.0.4 (Approximate block regression). Consider the problem

$$\arg\min_{x \in \mathbb{R}^d} \|Ax - b\|_2^2$$

where $A = \begin{bmatrix} A_1; A_2 \end{bmatrix}$ and $b = \begin{bmatrix} b_1; b_2 \end{bmatrix}$ are vertically stacked and $A_1 \in \mathbb{R}^{n_1 \times d}$, $A_2 \in \mathbb{R}^{n_2 \times d}$, $b_1 \in \mathbb{R}^{n_1}$, $b_2 \in \mathbb{R}^{n_2}$. Let $p \in [0, 1]^{n_1}$ be a $\beta$-overestimate for the leverage score distribution of $A_1$. Let $s \geq 1680d \ln(40d) / (\beta \varepsilon)$ and let $S$ be the output of SampleRows($A_1, s, p$). If

$$\tilde{x}^* = \arg\min_{x \in \mathbb{R}^d} \left( \|S(A_1 x - b_1)\|_2^2 + \|A_2 x - b_2\|_2^2 \right),$$

then, with probability at least $9/10$, we have

$$\|A\tilde{x}^* - b\|_2^2 \leq (1 + \varepsilon) \min_{x \in \mathbb{R}^d} \|Ax - b\|_2^2.$$  

We defer the proofs of these results to Section 4.7 — we essentially follow the outline and proofs of [176] (originally written in [152, Appendix B]). The key idea behind Lemma 4.3.0.4 is that leverage scores do not increase if rows are appended to the matrix. This then allows us to prove a sketched submatrix version of [177, Lemma 8] for approximate matrix multiplication and satisfy the structural conditions for approximate least squares in [178]. One consequence is that we can “sketch and solve” ridge regression, which was shown in [179, Theorem 1] and [180, Theorem 2].

Corollary 4.3.0.5. For any $A \in \mathbb{R}^{n \times d}$, $b \in \mathbb{R}^d$, $\lambda \geq 0$, consider

$$\arg\min_{x \in \mathbb{R}^d} \left( \|Ax - b\|_2^2 + \lambda \|x\|_2^2 \right).$$

Let $s \geq 1680d \ln(40d) / (\beta \varepsilon)$ and $p \in [0, 1]^{n_1}$ be a $\beta$-overestimate for the leverage scores of $A$. If $S$ is the output of SampleRows($A, s, p$), then, with probability at least $9/10$, the
sketched solution

\[ \tilde{x}^* = \arg \min_{x \in \mathbb{R}^d} \left( \| S (A x - b) \|_2^2 + \lambda \| x \|_2^2 \right) \]

gives a \((1 + \varepsilon)\)-approximation to the original problem.

**Remark 4.3.0.6.** The success probability of the sketch can be boosted from \(9/10\) to \(1 - \delta\) by sampling a factor of \(O(\log(1/\delta))\) more rows. See the discussion in [181, Section 2] about matrix concentration bounds for more details.

### 4.4 Kronecker Regression

Now we describe the key ingredients that allow us to design an approximate Kronecker regression algorithm whose running time is *subquadratic* in the number of columns in the design matrix.

1. The leverage score distribution of a Kronecker product matrix \(K = A^{(1)} \otimes \cdots \otimes A^{(N)}\) is a *product distribution* of the leverage score distributions of its factor matrices. Therefore, we can sample rows of \(K\) from \(\ell(K)\) with replacement in \(\tilde{O}(N)\) time after a preprocessing step.

2. The normal matrix \(K^T K + \lambda I\) in the ridge regression problem \(\min_x \| K x - b \|_2^2 + \lambda \| x \|_2^2\) is a \(O(1)\)-spectral approximation of the sketched matrix \((SK)^T SK + \lambda I\) by Lemma 4.3.0.3. Thus we can use Richardson iteration with \((K^T K + \lambda I)^+\) as the preconditioner to *solve the sketched instance*, which guarantees a \((1 + \varepsilon)\)-approximation. Using \((K^T K + \lambda I)^+\) as the preconditioner allows us to *heavily exploit the Kronecker structure* with fast matrix-vector multiplications.

3. At this point, *Kronecker matrix-vector multiplications* are still the bottleneck, so we partition the factor matrices into two groups by their number of columns and use our novel way of multiplying sparsified Kronecker product matrices as well as fast rectangular matrix multiplication to get a subquadratic running time.
This first result shows how $\lambda$-ridge leverage scores of a Kronecker product matrix decompose according to the SVDs of its factor matrices.

**Lemma 4.4.0.1.** Let $K = A^{(1)} \otimes A^{(2)} \otimes \cdots \otimes A^{(N)}$, where each factor matrix $A^{(n)} \in \mathbb{R}^{I_n \times R_n}$.

Let $(i_1, i_2, \ldots, i_N)$ be the natural row indexing of $K$ by its factors. Let the factor SVDs be $A^{(n)} = U^{(n)} \Sigma^{(n)} V^{(n)\top}$. For any $\lambda \geq 0$, the $\lambda$-ridge leverage scores of $K$ are

$$
\ell_{(i_1, \ldots, i_N)}(K) = \sum_{t \in T} \left( \frac{\prod_{n=1}^{N} \sigma_{i_n}^{2}(A^{(n)})}{\prod_{n=1}^{N} \sigma_{i_n}^{2}(A^{(n)})} + \lambda \left( \prod_{n=1}^{N} u_{i_n i_n}^{(n)} \right) \right)^{2},
$$

where the sum is over $T = [R_1] \times [R_2] \times \cdots \times [R_N]$. For statistical leverage scores, this simplifies to

$$
\ell_{(i_1, \ldots, i_N)}(K) = \prod_{n=1}^{N} \ell_{i_n}(A^{(n)}).
$$

**Proof.** For notational brevity, we prove the claim for $K = A \otimes B \otimes C$. The order-$N$ version follows by the same argument.

First, the mixed property property of Kronecker products implies that

$$
K\top K = (A\top A) \otimes (B\top B) \otimes (C\top C).
$$

Let $A = U_A \Sigma_A V_A\top$ be the SVD of $A$ such that $U_A \in \mathbb{R}^{I_1 \times I_1}$ and $V_A \in \mathbb{R}^{R_1 \times R_1}$. The orthogonality of $U_A$ implies that

$$
A\top A = V_A \Sigma_A^2 V_A\top,
$$

where $\Sigma_A^2$ denotes $\Sigma_A \Sigma_A$. Similarly, let $B = U_B \Sigma_B V_B\top$ and $C = U_C \Sigma_C V_C\top$. It follows from the mixed-product property that

$$
K\top K + \lambda I = \left( V_A \Sigma_A^2 V_A\top \right) \otimes \left( V_B \Sigma_B^2 V_B\top \right) \otimes \left( V_C \Sigma_C^2 V_C\top \right) + \lambda I
$$

$$
= (V_A \otimes V_B \otimes V_C) \left( \Sigma_A^2 \otimes \Sigma_B^2 \otimes \Sigma_C^2 \right) \left( V_A\top \otimes V_B\top \otimes V_C\top \right) + \lambda I
$$

$$
= (V_A \otimes V_B \otimes V_C) \left( \Sigma_A^2 \otimes \Sigma_B^2 \otimes \Sigma_C^2 + \lambda I \right) \left( V_A\top \otimes V_B\top \otimes V_C\top \right).$$

207
Since $(XY)^+ = Y^+X^+$ if $X$ or $Y$ is orthogonal, we have

$$(K^\top K + \lambda I)^+ = \left( (V_A \otimes V_B \otimes V_C) \left( \Sigma_A^2 \otimes \Sigma_B^2 \otimes \Sigma_C^2 + \lambda I \right) \left( V_A^\top \otimes V_B^\top \otimes V_C^\top \right) \right)^+$$

$= \left( V_A^\top \otimes V_B^\top \otimes V_C^\top \right)^+ \left( \Sigma_A^2 \otimes \Sigma_B^2 \otimes \Sigma_C^2 + \lambda I \right)^+ \left( V_A \otimes V_B \otimes V_C \right)^+$

$= (V_A \otimes V_B \otimes V_C) \left( \Sigma_A^2 \otimes \Sigma_B^2 \otimes \Sigma_C^2 + \lambda I \right)^+ \left( V_A^\top \otimes V_B^\top \otimes V_C^\top \right)$.

Next, observe that

$$K = (U_A \Sigma_A V_A^\top) \otimes (U_B \Sigma_B V_B^\top) \otimes (U_C \Sigma_C V_C^\top)$$

$$= (U_A \otimes U_B \otimes U_C) \left( \Sigma_A \otimes \Sigma_B \otimes \Sigma_C \right) \left( V_A^\top \otimes V_B^\top \otimes V_C^\top \right).$$

Putting everything together, the $\lambda$-ridge cross leverage scores can be expressed as

$$K (K^\top K + \lambda I)^+ K^\top = (U_A \otimes U_B \otimes U_C) \Lambda (U_A \otimes U_B \otimes U_C)^\top,$$  \hspace{1cm} (4.4.2)

where

$$\Lambda = \left( \Sigma_A \otimes \Sigma_B \otimes \Sigma_C \right) \left( \Sigma_A^2 \otimes \Sigma_B^2 \otimes \Sigma_C^2 + \lambda I \right)^+ \left( \Sigma_A \otimes \Sigma_B \otimes \Sigma_C \right).$$

(Equation 4.4.2) is the eigendecomposition of $K (K^\top K + \lambda I)^+ K^\top$. In particular, $\Lambda \in \mathbb{R}^{I_1 I_2 I_3 \times I_1 I_2 I_3}$ is a diagonal matrix of eigenvalues, where the $(i_1, i_2, i_3)$-th eigenvalue is

$$\lambda_{(i_1,i_2,i_3)} = \frac{\sigma_{i_1}^2 (A) \sigma_{i_2}^2 (B) \sigma_{i_3}^2 (C)}{\sigma_{i_1}^2 (A) \sigma_{i_2}^2 (B) \sigma_{i_3}^2 (C) + \lambda}.$$  \hspace{1cm} (4.4.3)

The value of $\ell_{(i_1,i_2,i_3),(j_1,j_2,j_3)}^1 (K)$ follows from the definition of cross $\lambda$-ridge leverage scores in (Equation 4.3.2).

Finally, the statistical leverage score property holds because setting $\lambda = 0$ gives an expression that is the product of the leverage scores of the factor matrices. \hfill \Box

This proof repeatedly uses the mixed-product property for Kronecker products and the
4.4.1 Iterative Methods

Now we state a result for the convergence rate of preconditioned Richardson iteration [182] using the notation $\|x\|_M^2 = x^T M x$.

**Lemma 4.4.1.1** (Preconditioned Richardson iteration). Let $M$ be any matrix such that $A^T A \leq M \leq \kappa \cdot A^T A$ for some $\kappa \geq 1$. Let $x^{(k+1)} = x^{(k)} - M \left( A^T A x^{(k)} - A^T b \right)$. Then,

$$\left\| x^{(k)} - x^* \right\|_M \leq (1 - 1/\kappa)^k \left\| x^{(0)} - x^* \right\|_M,$$

where $x^* = \arg \min_{x \in \mathbb{R}^d} \|Ax - b\|_2^2$.

**Remark 4.4.1.2.** The ridge regression algorithm in [147] is also based on sketching and preconditioned Richardson iteration. They consider short and wide matrices where $d \gg n$ and use the sketched normal matrix as the preconditioner to solve the original problem. One of our main technical contributions is to use the original normal matrix as the preconditioner to solve the sketched problem. Reversing this is advantageous because computing the pseduoinverse and matrix-vector products with the original Kronecker matrix is substantially less expensive due to its Kronecker structure. However, this still motivates the need for faster Kronecker matrix-vector multiplications.

4.4.2 Fast Kronecker-Matrix Multiplication

The next result is a simple but useful observation about extracting the rightmost factor matrix from the Kronecker product and recursively computing a new less expensive Kronecker-matrix multiplication.

**Lemma 4.4.2.1.** Let $A^{(n)} \in \mathbb{R}^{I_n \times J_n}$, for $n \in [N]$, and $B \in \mathbb{R}^{J_1 \cdots J_N \times K}$. There is an algorithm $\text{KronMatMul}([A^{(1)}, \ldots, A^{(N)}], B)$ that computes $\left( A^{(1)} \otimes A^{(2)} \otimes \cdots \otimes A^{(N)} \right) B \in \mathbb{R}^{(I_1 \cdots I_N) \times K}$ in $O \left( K \sum_{n=1}^{N} J_1 \cdots J_n I_n \cdots I_N \right)$ time.
Proof. We prove the claim by induction on $N$. Our approach will be to show that we can extract the rightmost factor matrix out of the Kronecker product and solve a smaller instance recursively.

If $N = 1$, this is a standard instance of matrix-matrix multiplication that takes $O(I_1J_1K)$ time. Now let

$$X = A^{(1)} \otimes \cdots \otimes A^{(N)} \in \mathbb{R}^{P \times Q}$$

and

$$Y = A^{(N+1)} \in \mathbb{R}^{R \times S}$$

Let $b \in \mathbb{R}^{QS}$ be an arbitrary column of $B$. We compute each of these $K$ matrix-vector products separately. Now we show how to efficiently compute $c = (X \otimes Y)b \in \mathbb{R}^{PR}$. The entry in $c$ at the canonical index $(p, r)$ is

$$c_{pr} = (x_{p, :} \otimes y_{r, :})b.$$

Writing this out, we have

$$c_{pr} = \sum_{q=1}^{Q} \sum_{s=1}^{S} x_{p, q} y_{r, s} b_{qs}$$

$$= \sum_{q=1}^{Q} x_{p, q} \sum_{s=1}^{S} y_{r, s} b_{qs}.$$

Therefore, for each $(q, r)$ we can precompute

$$z_{q, r} = \sum_{s=1}^{S} y_{r, s} b_{qs}.$$

Computing all of $Z \in \mathbb{R}^{Q \times R}$ takes $OQRS$ time. Now that we have $Z$, we can compute the
output e as:

$$c_{pr} = \sum_{q=1}^{Q} x_{p,q} \sum_{s=1}^{S} y_{r,s} b_{qs}$$

$$= \sum_{q=1}^{O} x_{p,q} z_{q,r}.$$ 

Therefore, we can write a natural matricized version of e as

$$C = XZ \in \mathbb{R}^{P \times R}.$$ 

This matrix C can be computed recursively since X is a Kronecker product. Translating back to the original dimensions as stated in the lemma, we have $P = I_1 \cdots I_N$, $Q = J_1 \cdots J_N$, $R = I_{N+1}$, and $S = J_{N+1}$. Computing Z takes time

$$O(QRS) = O(J_1 \cdots J_N I_{N+1} J_{N+1}) = O(J_1 \cdots J_{N+1} I_{N+1}).$$

By induction, the recursive solve for $XZ$ takes time

$$O \left( I_{N+1} \sum_{n=1}^{N} J_1 \cdots J_n I_n \cdots I_N \right).$$

Adding the two running times together and accounting for all $K$ columns of B gives us a total running time of

$$O \left( K \left( J_1 \cdots J_{N+1} I_{N+1} + I_{N+1} \sum_{n=1}^{N} J_1 \cdots J_n I_n \cdots I_N \right) \right) = O \left( K \sum_{n=1}^{N+1} J_1 \cdots J_n I_n \cdots I_{N+1} \right),$$

which completes the proof.  

We need the following theorem to prove the next theorem.
Lemma 4.4.2.2 ([183]). For a matrix \( C = [c_1, \ldots, c_q] \in \mathbb{R}^{p \times q} \), let

\[
\text{vec}(C) = \begin{bmatrix} c_1 \\ \vdots \\ c_q \end{bmatrix} \in \mathbb{R}^{pq}.
\]

Let \( A \in \mathbb{R}^{m \times q} \) and \( B \in \mathbb{R}^{n \times p} \). Then, \( \text{vec}(BCA^\top) = (A \otimes B) \text{vec}(C) \).

Theorem 4.4.2.3. Let \( A^{(1)} \in \mathbb{R}^{R_1 \times R_1}, \ldots, A^{(N)} \in \mathbb{R}^{R_N \times R_N} \) and \( c \in \mathbb{R}^{R_1 \cdots R_N} \). Let \( R = \prod_{n=1}^{N} R_n \). Then \( (A^{(1)} \otimes \cdots \otimes A^{(N)})c \) can be computed in time \( O(R \sum_{n=1}^{N} R_n) \).

Proof. Let \( C \) be an \( R_N \times (R_1 \cdots R_{N-1}) \) matrix such that \( c = \text{vec}(C) \) (see Lemma 4.4.2.2). For each \( n \in [N] \), let \( I_n \) be the identity matrix of size \( R_n \times R_n \). Then by Lemma 4.4.2.2,

\[
\left( A^{(1)} \otimes \cdots \otimes A^{(N)} \right) c = \left( A^{(1)} \otimes \cdots \otimes A^{(N)} \right) \text{vec}(C) \\
= \text{vec} \left( I_N A^{(N)} C \left( A^{(1)} \otimes \cdots \otimes A^{(N-1)} \right)^\top \right) \\
= \left( A^{(1)} \otimes \cdots \otimes A^{(N-1)} \otimes I_N \right) \text{vec} \left( A^{(N)} C \right). \tag{4.4.4}
\]

Since \( A^{(N)} \) is \( R_N \times R_N \) and \( C \) is \( R_N \times (R_1 \cdots R_{N-1}) \), \( A^{(N)} C \) can be computed in time \( O(R_N R) \).

Now note that although Kronecker product is not commutative, \( A \otimes B \) and \( B \otimes A \) are permutation equivalent, i.e., there are permutation matrices that transform one to the other. Therefore, instead of computing \( (A^{(1)} \otimes \cdots \otimes A^{(N-1)} \otimes I_N) \text{vec}(A^{(N)} C) \), we can compute \( (I_N \otimes A^{(1)} \otimes \cdots \otimes A^{(N-1)}) c_1 \), where \( c_1 \) is a permutation of \( \text{vec}(A^{(N)} C) \). We proceed with this multiplication and use a technique similar to (Equation 4.4.4), which results in a cost of \( O(R_{N-1} R) \). Continue until all the matrices in the Kronecker part are the identity. Then we can return a permutation of the final vector since the identity multiplied by a vector is the vector itself.

The following theorem is more sophisticated. We write the statement in terms of rect-
angular matrix multiplication time $MM(a, b, c)$, which is the time to multiply an $a \times b$ matrix by a $b \times c$ matrix.

**Theorem 4.4.2.4.** Let $A^{(n)} \in \mathbb{R}^{I_n \times R_n}$, for $n \in [N]$, $I = I_1 \cdots I_N$, $R = R_1 \cdots R_N$, $b \in \mathbb{R}^I$, $c \in \mathbb{R}^R$, and $S \in \mathbb{R}^{I \times I}$ be a diagonal matrix with $O(Re^{-1})$ nonzeros. The vectors

$\left( A^{(1)} \otimes \cdots \otimes A^{(N)} \right)^\top Sb \quad \text{and} \quad S \left( A^{(1)} \otimes \cdots \otimes A^{(N)} \right) c$

can be computed in time $\tilde{O} \left( \min_{T \subseteq [N]} MM \left( \prod_{n \in T} R_n, R_\epsilon^{-1}, \prod_{n \notin T} R_n \right) \right)$.

**Proof.** First observe that while the Kronecker product is not commutative, $A \otimes B$ and $B \otimes A$ are permutation equivalent, i.e., there are permutation matrices that transform one to the other. Therefore, without loss of generality we assume the minimizer

$$\arg \min_{T \subseteq [N]} MM(\prod_{n \in T} R_n, R_\epsilon^{-1}, \prod_{n \notin T} R_n)$$

is the set $[k]$ where $1 \leq k \leq N$. For any diagonal matrix $S$, let $S$ be the set corresponding to the indices of the nonzero entries of $S$ and let $I_S$ be a diagonal matrix where an entry is equal to one if its index is in $S$ and it is zero otherwise.

Note that because $S$ is an $(I_1 \cdots I_N) \times (I_1 \cdots I_N)$ matrix, each element of $S$ (i.e., nonzero of $S$) corresponds to a tuple $(i_1, \ldots, i_N) \in [I_1] \times \cdots \times [I_N]$. Let

$$S_1 = \{ (i_1, \ldots, i_k) : \exists i_{k+1} \in [I_{k+1}], \ldots, i_N \in [I_N] \text{ such that } (i_1, \ldots, i_N) \in S \},$$

$$S_2 = \{ (i_{k+1}, \ldots, i_N) : \exists i_1 \in [I_1], \ldots, i_k \in [I_k] \text{ such that } (i_1, \ldots, i_N) \in S \}.$$

Let $B_S$ be an $(I_1 \cdots I_k) \times (I_{k+1} \cdots I_N)$ matrix such that $Sb = \text{vec}(B_S)$ (see Lemma 4.4.2.2). Then by Lemma 4.4.2.2, we have

$$\left( A^{(1)} \otimes \cdots \otimes A^{(N)} \right)^\top Sb = \left( A^{(1)} \otimes \cdots \otimes A^{(N)} \right)^\top I_S Sb$$

213
Moreover, we have columns.\(\text{vec}(\mathbf{B}_S)\)\).

Note that the number of nonzeros in \(\mathbf{B}_S\) is equal to the number of nonzeros in \(\mathbf{S}\), which is \(O(R/\epsilon)\). Therefore, \((\mathbf{A}^{(k+1)} \otimes \cdots \otimes \mathbf{A}^{(N)})^\top \mathbf{I}_{\mathcal{S}_2}\mathbf{B}_S\) can be computed in \(O(R/\epsilon \prod_{n=k+1}^N R_n)\) time because \((\mathbf{A}^{(k+1)} \otimes \cdots \otimes \mathbf{A}^{(N)})^\top \mathbf{I}_{\mathcal{S}_2}\) has \(\prod_{n=k+1}^N R_n\) rows and \(\mathbf{B}_S\) has \(O(R/\epsilon)\) nonzero entries. Moreover,

\[
O\left(\frac{R}{\epsilon} \prod_{n=k+1}^N R_n\right) = O\left(MM\left(1, \frac{R}{\epsilon}, \prod_{n=k+1}^N R_n\right)\right) = O\left(MM\left(k \prod_{n=1}^k R_n, \frac{R}{\epsilon}, \prod_{n=k+1}^N R_n\right)\right).
\]

Now, note that \(|S_1| \leq |S| = O(R/\epsilon)\). Thus, multiplying \((\mathbf{A}^{(k+1)} \otimes \cdots \otimes \mathbf{A}^{(N)})^\top \mathbf{I}_{\mathcal{S}_2}\mathbf{B}_S\) with \((\mathbf{I}_{\mathcal{S}_1}(\mathbf{A}^{(1)} \otimes \cdots \otimes \mathbf{A}^{(k)}))\) can be done in time \(O(MM(\prod_{n=1}^k R_n, R/\epsilon, \prod_{n=k+1}^N R_n))\) because \((\mathbf{A}^{(k+1)} \otimes \cdots \otimes \mathbf{A}^{(N)})^\top \mathbf{I}_{\mathcal{S}_2}\mathbf{B}_S\) has \(\prod_{n=k+1}^N R_n\) rows and \(\mathbf{A}^{(1)} \otimes \cdots \otimes \mathbf{A}^{(k)}\) has \(\prod_{n=1}^k R_n\) columns.

Now let \(\mathbf{C}\) be an \((R_{k+1} \cdots R_N) \times (R_1 \cdots R_k)\) matrix such that \(\mathbf{c} = \text{vec}(\mathbf{C})\). Then we have

\[
\mathbf{S}(\mathbf{A}^{(1)} \otimes \cdots \otimes \mathbf{A}^{(N)})\mathbf{c} = \mathbf{S}\mathbf{I}_{\mathcal{S}}(\mathbf{A}^{(1)} \otimes \cdots \otimes \mathbf{A}^{(N)})\mathbf{c}
\]

\[
= \mathbf{S}(\mathbf{I}_{\mathcal{S}_1} \otimes \mathbf{I}_{\mathcal{S}_2}) (\mathbf{A}^{(1)} \otimes \cdots \otimes \mathbf{A}^{(N)})\mathbf{c}
\]

\[
= \mathbf{S}(\mathbf{I}_{\mathcal{S}_1}(\mathbf{A}^{(1)} \otimes \cdots \otimes \mathbf{A}^{(k)})) \otimes (\mathbf{I}_{\mathcal{S}_2}(\mathbf{A}^{(k+1)} \otimes \cdots \otimes \mathbf{A}^{(N)}))\text{vec}(\mathbf{C})
\]

\[
= \mathbf{S}\text{vec}((\mathbf{I}_{\mathcal{S}_2}(\mathbf{A}^{(k+1)} \otimes \cdots \otimes \mathbf{A}^{(N)}))\mathbf{C}((\mathbf{A}^{(1)} \otimes \cdots \otimes \mathbf{A}^{(k)})^\top \mathbf{I}_{\mathcal{S}_1}))
\]

We have \(|S_2| \leq |S| = O(R/\epsilon)\). Therefore, \(\mathbf{I}_{\mathcal{S}_2}(\mathbf{A}_{k+1} \cdots \otimes \mathbf{A}_N)\) has \(O(R/\epsilon)\) nonzero entries. Moreover, \(\mathbf{C}\) is an \((R_{k+1} \cdots R_N) \times (R_1 \cdots R_k)\) matrix. Hence, \((\mathbf{I}_{\mathcal{S}_2}(\mathbf{A}^{(k+1)} \otimes \cdots \otimes \mathbf{A}^{(N)}))\mathbf{C}\)
can be computed in time
\[ O(\text{MM}(\frac{R}{\epsilon}, \prod_{n=k+1}^{N} R_n, \prod_{n=1}^{k} R_n)) = O(\text{MM}(\prod_{n=1}^{N} \frac{R}{\epsilon}, \prod_{n=k+1}^{N} R_n)). \]

Observe that we do not need to compute all entries of
\[ \left( \mathbf{I}_{S_2}(\mathbf{A}^{(k+1)} \otimes \cdots \otimes \mathbf{A}^{(N)}) \right) \mathbf{C}(\left( \mathbf{A}^{(1)} \otimes \cdots \otimes \mathbf{A}^{(k)} \right)^\top \mathbf{I}_{S_1}). \]

Instead, we only need to compute entries corresponding to nonzero entries of \( S \). Computing each such entry takes \( O(\prod_{n=1}^{k} R_n) \) time because \(\left( \mathbf{I}_{S_2}(\mathbf{A}^{(k+1)} \otimes \cdots \otimes \mathbf{A}^{(N)}) \right) \mathbf{C}(\left( \mathbf{A}^{(1)} \otimes \cdots \otimes \mathbf{A}^{(k)} \right)^\top \mathbf{I}_{S_1}) \) have \( \prod_{n=1}^{k} R_n \) columns and rows, respectively. Moreover, the number of nonzeros is \( S \) is \( \tilde{O}(R/\epsilon) \). Therefore, computing all entries of \( \left( \mathbf{I}_{S_2}(\mathbf{A}^{(k+1)} \otimes \cdots \otimes \mathbf{A}^{(N)}) \right) \mathbf{C}(\left( \mathbf{A}^{(1)} \otimes \cdots \otimes \mathbf{A}^{(k)} \right)^\top \mathbf{I}_{S_1}) \) that correspond to nonzero entries of \( S \) takes time
\[ O\left( \frac{R}{\epsilon} \prod_{n=1}^{k} R_n \right) = O\left( \text{MM}\left( \prod_{n=1}^{k} \frac{R}{\epsilon}, 1 \right) \right) = O\left( \text{MM}\left( \prod_{n=1}^{k} R_n, \frac{R}{\epsilon}, \prod_{n=k+1}^{N} R_n \right) \right). \]

The core idea behind Theorem 4.4.2.4 is that the factor matrices can be partitioned into two groups to achieve a good “column-product” balance, i.e.,
\[ \min_{T \subseteq [N]} \max_{n \in T} \min_{n \notin T} R_n \]
is close to \( \sqrt{R} \). Then we use the fact that \( \text{NNZ}(S) = \tilde{O}(R\epsilon^{-1}) \) with a sparsity-aware KronMatMul to solve each part of this partition separately, and combine them with fast rectangular matrix multiplication. If we achieve perfect balance, the running time is \( \tilde{O}(R^{1.626} \epsilon^{-1}) \) using results of [104], which are explained in detail in [90, Appendix C]. If one of these two factor matrix groups has at most 0.9 of the “column-product mass,” the running time is \( \tilde{O}(R^{1.9} \epsilon^{-1}) \).
4.4.3 Main Algorithm

We are now ready to present our main algorithm for solving approximate Kronecker regression. The main ideas behind Algorithm 17 are as the following. First, we do not compute the pseudoinverse $\tilde{K}^+$ but instead use iterative Richardson iteration (Lemma 4.4.1.1), which allows us avoid a $\tilde{O}(R\omega\epsilon^{-1})$ running time. This technique by itself, however, only allows us to reduce the running time to $\tilde{O}(R^2\epsilon^{-1})$ since all of the matrix-vector products (e.g., $\tilde{K}^T\tilde{b}$, $\tilde{K}x$, and multiplication against $M^+$) naively take $\Omega(R^2)$ time. To achieve subquadratic time, we need three more ideas: (1) compute an approximate SVD of each Gram matrix $A^{(n)}^TA^{(n)}$ in order to construct the decomposed preconditioner $M^+$; (2) use fast Kronecker-vector multiplication (e.g., Lemma 4.4.2.1) to exploit the Kronecker structure of the decomposed preconditioner; (3) noting that Lemma 4.4.2.1 for the Kronecker-vector products $\tilde{K}^T\tilde{b}$ and $\tilde{K}^T(\tilde{K}x)$ is insufficient because the intermediate vectors can be large, we develop a novel multiplication algorithm in Theorem 4.4.2.4 that fully exploits the sparsity, Kronecker structure, and fast rectangular matrix multiplication of [104].

We need the following lemmas to prove the main result.

**Lemma 4.4.3.1** (Johnson–Lindenstrauss random projection [184, 185]). Let $x \in \mathbb{R}^d$. Assume the entries in $G \in \mathbb{R}^{r \times d}$ are sampled independently from $N(0, 1)$. Then,

$$\Pr\left(\left(1 - \epsilon\right)\|x\|_2^2 \leq \left\|\frac{1}{\sqrt{r}}Gx\right\|_2^2 \leq (1 + \epsilon)\|x\|_2^2\right) \geq 1 - 2e^{-\left(\epsilon^2 - \epsilon^3\right)r/4}.$$ 

**Lemma 4.4.3.2.** Let $A \in \mathbb{R}^{n \times d}$ and $0 < \epsilon \leq 1/4$. Given $\tilde{A} \in \mathbb{R}^{k \times d}$ and $N = \tilde{A}^T\tilde{A} \in \mathbb{R}^{d \times d}$ such that

$$A^TA \preceq \tilde{A}^T\tilde{A} \preceq (1 + \epsilon/4)A^TA,$$

with high probability, all leverage scores of $A$ can be computed to a $(1 + \epsilon/2)$ approximation in $\tilde{O}(\text{nnz}(A) + kd + d^\omega)$ time.

*Proof.* Let $M = (1 + \epsilon/4)(\tilde{A}^T\tilde{A})^{-1}$. It follows that $(A^TA)^{-1} \preceq M \preceq (1 + \epsilon/4)(A^TA)^{-1}$.  

216
Hence, for any \( x \in \mathbb{R}^d \), we have
\[
x^\top (A^\top A)^{-1} x \leq x^\top M x \leq (1 + \varepsilon/4) x^\top (A^\top A)^{-1} x.
\]

Now note that \( M = MM^{-1} = \frac{1}{1 + \varepsilon/4} M \tilde{A}^\top \tilde{A} M \). Hence,
\[
x^\top M x = \frac{1}{1 + \varepsilon/4} x^\top \tilde{A}^\top \tilde{A} M x = \frac{1}{1 + \varepsilon/4} \| \tilde{A} M x \|^2.
\]

Using Lemma 4.4.3.1 with \( \varepsilon/20 \) and \( r = O(\log n) \), we can compute a random matrix \( G \) such that, with high probability, for all \( a_i \), we have
\[
\| \tilde{A} Ma_i \|^2 \leq \frac{1}{1 - \varepsilon/20} \| G \tilde{A} Ma_i \|^2 \leq \frac{1 + \varepsilon/20}{1 - \varepsilon/20} \| \tilde{A} Ma_i \|^2 \leq (1 + \varepsilon/6) \| \tilde{A} Ma_i \|^2.
\]

Combining the above, we have
\[
a_i^\top (A^\top A)^{-1} a_i \leq a_i^\top Ma_i = \frac{1}{1 + \varepsilon/4} \| \tilde{A} Ma_i \|^2 \leq \frac{1}{(1 + \varepsilon/4)(1 - \varepsilon/20)} \| G \tilde{A} Ma_i \|^2,
\]
and
\[
\frac{1}{(1 + \varepsilon/4)(1 - \varepsilon/20)} \| G \tilde{A} Ma_i \|^2 \leq a_i^\top Ma_i \leq (1 + \varepsilon/4) a_i^\top (A^\top A)^{-1} a_i.
\]

Therefore, \( \frac{1}{(1 + \varepsilon/4)(1 - \varepsilon/20)} \| G \tilde{A} Ma_i \|^2 \) is a \( 1 + \varepsilon/4 \leq 1 + \varepsilon/2 \) approximation of the leverage score of \( a_i \).

Lastly, we discuss the running time. Note that given \( N \), we can compute \( M \) in \( \tilde{O}(d^\omega) \) time. Moreover, since \( G \) has \( \tilde{O}(1) \) rows, \( G \tilde{A} \) and \( G \tilde{A} M \) can be computed in \( \tilde{O}(kd + d^2) \) time. Finally, given \( G \tilde{A} M \), we can compute \( G \tilde{A} Ma_i \), for all \( i \in [n] \), in \( \tilde{O}(\text{nnz}(A)) \).

**Theorem 4.4.3.3.** For any Kronecker product matrix \( K = A^{(1)} \otimes \cdots \otimes A^{(N)} \in \mathbb{R}^{I_1 \cdots I_N \times R_1 \cdots R_N} \), \( b \in \mathbb{R}^{I_1 \cdots I_N} \), \( \lambda \geq 0 \), \( \varepsilon \in (0, 1/4] \), and \( \delta > 0 \), FastKroneckerRegression returns
Proof. By [40, Lemma 8], we can compute a $(1+\varepsilon/N)$-spectral approximation $\tilde{A}^{(n)}$ of $A^{(n)}$, with $\tilde{O}(R_n N^2 \varepsilon^{-2})$ rows, in $\tilde{O}(\text{nnz}(A^{(n)}) + R_n^\omega N^2 \varepsilon^{-2})$ time. Given $\tilde{A}^{(n)}$, we can compute $\tilde{A}^{(n)^\top} \tilde{A}^{(n)}$ in $\tilde{O}(R_n^\omega N^2 \varepsilon^{-2})$ time. Finally, given $\tilde{A}^{(n)^\top} \tilde{A}^{(n)} \in \mathbb{R}^{R_n \times R_n}$, we can compute its
SVD in $\tilde{O}(R_n^\omega)$ time.

Given $\tilde{A}^{(n)}$ and $\tilde{A}^{(n)^\top}\tilde{A}^{(n)}$, by Lemma 4.4.3.2, it takes $\tilde{O}(\text{nnz}(A^{(n)}) + R_n^\omega N^2 \epsilon^{-2})$ time to compute the (approximate) leverage scores $\ell(A^{(n)})$. Compute the cumulative density function of each leverage score distribution in $O(I_n)$ time. This allows us to sample from the product distribution $\mathcal{P} = \ell(A^{(1)}) \otimes \cdots \otimes \ell(A^{(N)})$ in $\tilde{O}(N)$ by sampling each coordinate independently. Sampling from $\mathcal{P}$ is equivalent to sampling from $\ell(K)$ by Lemma 4.4.0.1.

Note that

$$
(1 + \log(1 + \epsilon/4)/N)^N \leq e^{\log(1+\epsilon/4)/N} = e^{\log(1+\epsilon/4)} = 1 + \epsilon/4.
$$

Therefore, because for all $n \in [N]$, we have

$$
A^{(n)^\top}A^{(n)} \preceq \tilde{A}^{(n)^\top}\tilde{A}^{(n)} \preceq (1 + \log(1 + \epsilon/4)/N)A^{(n)^\top}A^{(n)},
$$

it follows that

$$
(A^{(1)^\top}A^{(1)}) \otimes \cdots \otimes (A^{(N)^\top}A^{(N)}) \preceq (\tilde{A}^{(1)^\top}\tilde{A}^{(1)}) \otimes \cdots \otimes (\tilde{A}^{(N)^\top}\tilde{A}^{(N)})
\preceq (1 + \epsilon/4)(A^{(1)^\top}A^{(1)}) \otimes \cdots \otimes (A^{(N)^\top}A^{(N)}).
$$

Thus, the approximate leverage scores we get in Algorithm 17 for $A^{(1)} \otimes \cdots \otimes A^{(N)}$ are within a factor of $(1 + \epsilon/4)$ of the true leverage scores.

Therefore, our preconditioner given by the SVD, $V_K (\Sigma_K^\top \Sigma_K + \lambda I_R)^+ V_K^\top$, is a spectral approximation of $(K^\top K + \lambda I_R)^+$. More specifically,

$$
\tilde{K}^\top \tilde{K} + \lambda I \preceq \frac{1}{1 - \sqrt{\epsilon}} (K^\top K + \lambda I_R) \preceq \frac{1}{1 - \sqrt{\epsilon}} V_K (\Sigma_K^\top \Sigma_K + \lambda I_R)^+ V_K^\top 
\preceq \frac{1 + \epsilon/4}{1 - \sqrt{\epsilon}} (K^\top K + \lambda I_R) \preceq \frac{(1 + \epsilon/4)(1 + \sqrt{\epsilon})}{1 - \sqrt{\epsilon}} (\tilde{K}^\top \tilde{K} + \lambda I).
$$

Therefore, by $O(\log(1/\epsilon))$ iterations of Richardson (Lemma 4.4.1.1), we converge to the
desired accuracy. Finally, note that each iteration of Richardson can be done in time

$$O(\min_{S \subseteq [N]} MM\left(\prod_{n \in S} R_n, R\varepsilon^{-1}, \prod_{n \in [N] \setminus S} R_n\right)),$$

using our novel procedure for sparse Kronecker-matrix multiplication (Theorem 4.4.2.4), KronMatMul (Lemma 4.4.2.1), and the structure of the preconditioner, which is a diagonal matrix multiplied from left and right by matrices with Kronecker structure. □

### 4.5 Applications to Low-Rank Tucker Decomposition

Now we apply our fast Kronecker regression algorithm to TuckerALS and prove Theorem 4.1.0.2. We list the running times of different factor matrix and core update algorithms in Table 4.1. Our result for factor matrix update is analyzed in this section.

**Core Tensor Update.** The core update running time in Theorem 4.1.0.2 is a direct consequence of our algorithm for fast Kronecker regression in Theorem 4.4.3.3. The only difference is that we avoid recomputing the SVD and Gram matrix of each factor since these are computed at the end of each factor matrix update and stored for future use.

**Factor Matrix Update.** The factor matrix updates require more work because of the $G_{(n)}^T y$ term in Line 8 of TuckerALS. To overcome this, we substitute variables and re-cast each factor update as an equality-constrained Kronecker regression problem with an appended low-rank block to account for the L2 regularization of the original variables. To support this new low-rank block, we use the Woodbury matrix identity to extend the technique of using Richardson iterations with fast Kronecker matrix-vector multiplication for solving sketched regression instances.

The next result formalizes this substitution and reduces the problem to block Kronecker regression with a subspace constraint. This result relies on the fact that the least squares solution to $\|Mx - z\|_2^2$ with minimum norm is $M^+z$. 220
Lemma 4.5.0.1. Let $A \in \mathbb{R}^{n \times m}$, $M \in \mathbb{R}^{m \times d}$, $b \in \mathbb{R}^{n}$, and $\lambda \geq 0$. For any ridge regression problem of the form $\arg \min_{x \in \mathbb{R}^{d}} \|AMx - b\|_2^2 + \lambda \|x\|_2^2$, we can solve $z_{opt} = \arg \min_{Nz=0} \|Az - b\|_2^2 + \lambda \|Mz\|_2^2$, where $N = I_m - MM^+$, and return vector $M^+z_{opt}$ instead.

Proof. Let $z = Mx \in \mathbb{R}^{m}$. For any $x \in \mathbb{R}^{d}$, $z$ is in the column space of $M$ and hence orthogonal to any vector in the left null space of $M$. Therefore, we can optimize over $z \in \mathbb{R}^{m}$ subject to $Nz = 0$ instead because for any $x \in \mathbb{R}^{d}$, $NMx = (I_m - MM^+)Mx = (M - M)x = 0$. Using this substitution, we can also replace the term $\lambda \|x\|_2^2$ by $\lambda \|Mz\|_2^2$ because for any $z$, the least squares solution to $z = Mx$ with minimum norm is $M^+z$ [109].

To solve this constrained regression problem, we can add a scaled version of the constraint matrix $N$ as a block to the approximate regression problem and take the projection of the resulting solution.

Lemma 4.5.0.2 (Approximate equality-constrained regression). Let $M \in \mathbb{R}^{n \times d}$, $N \in \mathbb{R}^{m \times d}$, $b \in \mathbb{R}^{n}$, and $0 < \epsilon < 1/3$. To solve $\min_{Nx=0} \|Mx - b\|_2^2$ to a $(1 + \epsilon)$-approximation, it suffices to solve

$$
\min_{x \in \mathbb{R}^{d}} \left\| \begin{bmatrix} M \\ \sqrt{w}N \end{bmatrix} x - \begin{bmatrix} b \\ 0 \end{bmatrix} \right\|_2^2
$$

to a $(1 + \epsilon/3)$-approximation with $w \geq (1 + 12/\epsilon) \|MN^+\|_2^2$.

Proof. First note that for any $w \geq 0$, we have

$$
\min_{x \in \mathbb{R}^{d}} \left\| \begin{bmatrix} M \\ \sqrt{w}N \end{bmatrix} x - \begin{bmatrix} b \\ 0 \end{bmatrix} \right\|_2^2
\leq \min_{Nx=0} \left\| \begin{bmatrix} M \\ \sqrt{w}N \end{bmatrix} x - \begin{bmatrix} b \\ 0 \end{bmatrix} \right\|_2^2
= \min_{Nx=0} \|Mx - b\|_2^2. \tag{4.5.1}
$$

Suppose $\hat{x} \in \mathbb{R}^{d}$ such that

$$
\left\| \begin{bmatrix} M \\ \sqrt{w}N \end{bmatrix} \hat{x} - \begin{bmatrix} b \\ 0 \end{bmatrix} \right\|_2 \leq (1 + \epsilon/3) \min_{x \in \mathbb{R}^{d}} \left\| \begin{bmatrix} M \\ \sqrt{w}N \end{bmatrix} x - \begin{bmatrix} b \\ 0 \end{bmatrix} \right\|_2. \tag{4.5.2}
$$

221
Let $z = (I - N^*N)\hat{x}$. It follows that $Nz = 0$ because $N = NN^*N$. Therefore,

$$\left\| \frac{M}{\sqrt{w}N} z - \begin{bmatrix} b \\ 0 \end{bmatrix} \right\|_2^2 = \left\| \frac{M}{\sqrt{w}N} (I - N^*N)\hat{x} - \begin{bmatrix} b \\ 0 \end{bmatrix} \right\|_2^2 = \left\| M(I - N^*N)\hat{x} - b \right\|_2^2.$$  

By the triangle inequality,

$$\left\| M(I - N^*N)\hat{x} - b \right\|_2 \leq \left\| M\hat{x} - b \right\|_2 + \left\| \frac{M^*N\hat{x}}{\sqrt{w}} \right\|_2.$$  

Therefore,

$$\left\| M(I - N^*N)\hat{x} - b \right\|_2^2 \leq \left\| M\hat{x} - b \right\|_2^2 + \left\| \frac{M^*N\hat{x}}{\sqrt{w}} \right\|_2^2 + 2 \left\| M\hat{x} - b \right\|_2 \left\| \frac{M^*N\hat{x}}{\sqrt{w}} \right\|_2.$$  

Now we have two cases:

- Case 1: $2 \left\| \frac{M^*N\hat{x}}{\sqrt{w}} \right\|_2 \leq \frac{\epsilon}{3} \left\| M\hat{x} - b \right\|_2$,

- Case 2: $2 \left\| \frac{M^*N\hat{x}}{\sqrt{w}} \right\|_2 > \frac{\epsilon}{3} \left\| M\hat{x} - b \right\|_2$.

Note that by the consistency of operator norms, we have

$$\left\| \frac{M^*N\hat{x}}{\sqrt{w}} \right\|_2 \leq \left\| \frac{M^*N}{\sqrt{w}} \right\|_2 \left\| \hat{x} \right\|_2.$$  

Therefore, in the first case we have

$$\left\| M(I - N^*N)\hat{x} - b \right\|_2^2 \leq \left( 1 + \frac{\epsilon}{3} \right) \left\| M\hat{x} - b \right\|_2^2 + \left\| \frac{M^*N\hat{x}}{\sqrt{w}} \right\|_2^2 \leq \left( 1 + \frac{\epsilon}{3} \right) \left( \left\| M\hat{x} - b \right\|_2^2 + w \left\| \hat{x} \right\|_2^2 \right),$$  

where the last inequality follows from our choice of $w$. In the second case we have

$$\left\| M(I - N^*N)\hat{x} - b \right\|_2 \leq \left\| M\hat{x} - b \right\|_2^2 + \left( 1 + \frac{12}{\epsilon} \right) \left\| \frac{M^*N\hat{x}}{\sqrt{w}} \right\|_2^2.$$  

222
\[ \leq \|M\hat{x} - b\|_2^2 + w \|N\hat{x}\|_2^2. \]

Therefore, in both cases

\[ \|M(I - N^+N)\hat{x} - b\|_2^2 \leq \left(1 + \frac{\varepsilon}{3}\right) (\|M\hat{x} - b\|_2^2 + w \|N\hat{x}\|_2^2). \] (4.5.3)

Moreover, \(\varepsilon < 1/3\), so then \((1 + \varepsilon/3)^2 \leq 1 + \varepsilon\). Thus by (Equation 4.5.1), (Equation 4.5.2) and (Equation 4.5.3), we have

\[ \|Mz - b\|_2^2 = \|M(I - N^+N)\hat{x} - b\|_2^2 \leq (1 + \varepsilon) \min_{x \in \mathbb{R}^d} \left[ \begin{bmatrix} M \\ \sqrt{wN} \end{bmatrix} x - \begin{bmatrix} b \\ 0 \end{bmatrix} \right]\|_2^2 \]

\[ \leq (1 + \varepsilon) \min_{Nx=0} \|Mx - b\|_2^2. \]

Finally, note that \(Nz = 0\) and that \(z\) is a feasible solution. \(\square\)

Now, we explain how the reduction to an equality-constrained least squares problem in Lemma 4.5.0.1 applies to factor matrix updates in TuckerALS. For the factor matrix updates, we solve a regression problem of the form:

\[ \arg\min_{y \in \mathbb{R}^{R_n}} \left\| \left( A^{(1)} \otimes \cdots \otimes A^{(n-1)} \otimes A^{(n+1)} \otimes \cdots \otimes A^{(N)} \right) G_{(n)}^T y - b_{(i)} : \right\|_2^2 + \lambda \|y\|_2^2, \] (4.5.4)

where \(b_{(i)}\) is the \(i\)-th row of the mode-\(n\) unfolding of tensor \(\mathcal{X}\). Note that for any \(y\), \(G_{(n)}^T y\) is a vector in the column space of \(G_{(n)}^T\). Thus, \(G_{(n)}^T y\) is orthogonal to any vector in the left null space of \(G_{(n)}^T\). Let \(N\) be a matrix in which the rows are a basis for the left null space of \(G_{(n)}^T\). Then, solving the following is equivalent to solving (Equation 4.5.4):

\[ \min_{Nz=0} \left\| \left( A^{(1)} \otimes \cdots \otimes A^{(n-1)} \otimes A^{(n+1)} \otimes \cdots \otimes A^{(N)} \right) z - b_{(i)} : \right\|_2^2 + \lambda \left\| (G_{(n)}^T)^+ z \right\|_2^2, \] (4.5.5)

where \((G_{(n)}^T)^+\) is the pseudoinverse of \(G_{(n)}^T\).
To explain the $\left\| (G^\top_{(n)})^+ z \right\|_2^2$ term, consider a vector $y$ that under the transformation $G^\top_{(n)}$ goes to $z$, i.e., $G^\top_{(n)}y = z$. The set of solutions to this linear system is $(G^\top_{(n)})^+ z + (I - (G^\top_{(n)})^+ G^\top_{(n)})w$, for all $w$ by [186]. Moreover, $(G^\top_{(n)})^+ z$ is orthogonal to $(I - (G^\top_{(n)})^+ G^\top_{(n)})w$ because

\[
((G^\top_{(n)})^+ z)^\top (I - (G^\top_{(n)})^+ G^\top_{(n)}) w = ((G^\top_{(n)})^+ G^\top_{(n)} (G^\top_{(n)})^+ z)^\top (I - (G^\top_{(n)})^+ G^\top_{(n)}) w
= ((G^\top_{(n)})^+ z)^\top (G^\top_{(n)})^+ (I - (G^\top_{(n)})^+ G^\top_{(n)}) w
= ((G^\top_{(n)})^+ z)^\top (G^\top_{(n)})^+ (G^\top_{(n)} - G^\top_{(n)} (G^\top_{(n)})^+ G^\top_{(n)}) w
= 0.
\]

Therefore, by the Pythagorean theorem, we have

\[
\left\| (G^\top_{(n)})^+ z + (I - (G^\top_{(n)})^+ G^\top_{(n)}) w \right\|_2^2 = \left\| (G^\top_{(n)})^+ z \right\|_2^2 + \left\| (I - (G^\top_{(n)})^+ G^\top_{(n)}) w \right\|_2^2,
\]

so $\left\| (G^\top_{(n)})^+ z + (I - (G^\top_{(n)})^+ G^\top_{(n)}) w \right\|_2^2$ is minimized when $w = 0$ [109]. Thus, for all $y$ such that $G^\top_{(n)} y = z$, it follows that $(G^\top_{(n)})^+ z$ minimizes $\|y\|_2^2$ in (Equation 4.5.4), hence we can replace $y$ by $(G^\top_{(n)})^+ z$.

Note that $N = I - G^\top_{(n)} (G^\top_{(n)})^+$ works because for any $y$, by definition of pseudoinverse:

\[
(I - G^\top_{(n)} (G^\top_{(n)})^+) G^\top_{(n)} y = (G^\top_{(n)} - G^\top_{(n)} (G^\top_{(n)})^+ G^\top_{(n)}) y = 0.
\]

More generally, a vector $z$ is in the image of $G^\top_{(n)}$ if and only if $(I - G^\top_{(n)} (G^\top_{(n)})^+) z = 0$. This is an alternate formulation of Lemma 4.5.0.1 and leads to Algorithm 18.

**Theorem 4.5.0.3.** Let $\lambda \geq 0$, $\varepsilon \in (0, 1/3)$, $\delta > 0$. Then FastFactorMatrixUpdate algorithm updates $A_{(k)} \in \mathbb{R}^{I_k \times R_k}$ in TuckerALS with a $(1 + \varepsilon)$-approximation, with prob-
Algorithm 18: FastFactorMatrixUpdate

1. **Input:** Tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$, factors $A^{(n)} \in \mathbb{R}^{I_n \times R_n}$, core $G \in \mathbb{R}^{R_1 \times \cdots \times R_N}$, index $n$, error $\varepsilon$, probability $\delta$

2. Set $R_{\neq n} \leftarrow R_1 \cdots R_{n-1} R_{n+1} \cdots R_N$

3. $K = A^{(1)} \otimes \cdots \otimes A^{(n-1)} \otimes A^{(n+1)} \otimes \cdots \otimes A^{(N)}$

4. Set $B \leftarrow X^{(n)}$

5. Initialize product distribution data structure $\mathcal{P}$ to sample indices from $(\ell(A^{(1)}), \cdots, \ell(A^{(N)}))$

6. Let $N = I_{R_{\neq n}} - G_{(n)}^T (G_{(n)})^+$

7. Let $w \geq \left(1 + \frac{12}{\varepsilon} \right) \left\| \sqrt{\lambda} (G_{(n)}^T)^+ \right\|_2^2$ as in Lemma 4.5.0.2, and construct the operator

$$M^+ = \begin{pmatrix} K \\ \sqrt{\lambda} (G_{(n)}^T)^+ \end{pmatrix}^T \begin{pmatrix} K \\ \sqrt{\lambda} (G_{(n)}^T)^+ \end{pmatrix}^+$$

using the Woodbury identity in (Equation 4.5.6)

8. Set $s \leftarrow \lceil 1680R_{\neq n} \ln(40R_{\neq n}) \ln(I_n/\delta)/\varepsilon \rceil$

9. for $i = 1$ to $I_n$ do

10. Set $S \leftarrow \text{SampleRows}(K, s, \mathcal{P})$

11. Set $\tilde{K} \leftarrow SK$ and $\tilde{b} \leftarrow Sb_i^T$

12. Initialize $z \leftarrow 0_{R_{\neq n}}$

13. while has not converged do

14. Update

$$z \leftarrow z - (1 - \sqrt{\varepsilon}) M^+ \left( (\tilde{K}^T \tilde{K} + wI)z + \lambda G_{(n)}^+ (G_{(n)}^T)^+ z - wG_{(n)}^+ G_{(n)} z - \tilde{K}^T \tilde{b} \right)$$

using fast Kronecker-matrix multiplication

15. Update factor matrix row $a_{i:}^{(n)} \leftarrow z^T G_{(n)}^+$

ability at least $1 - \delta$, in time

$$\tilde{O} \left( I_k R_{\neq k}^2 \varepsilon^{-1} \log(1/\delta) + I_k R \sum_{n=1}^N R_n + R_k^\omega \varepsilon^{-2} \right).$$

**Proof.** Each factor row matrix update in TuckerALS (Algorithm 16) has the form

$$a_{i:}^{(n)} \leftarrow \arg \min_{y \in \mathbb{R}^{R_n}} \left( \left\| KG_{(n)}^T y - b_i^T \right\|_2^2 + \lambda \left\| y \right\|_2^2 \right).$$
Use Lemma 4.5.0.1 to reduce the factor matrix updates to solving the equality-constrained Kronecker regression problem

\[
\mathbf{z}_{\text{opt}} = \arg \min_{\mathbf{z}} \left( \| \mathbf{Kz} - \mathbf{b}_i^\top \|_2^2 + \lambda \left\| (\mathbf{G}_{(n)}^\top)_+ \mathbf{z} \right\|_2^2 \right).
\]

The correctness of this algorithm is analogous to the argument in the proof of Theorem 4.4.3.3, but now we have more sophisticated blocks in the data matrix and need to account for them.

We solve each row update independently. The construction of the sketched submatrix \( \tilde{\mathbf{K}} \) guarantees that \( (1 - \sqrt{\varepsilon})^{-1} \mathbf{M} \) is a 3-spectral approximation to the sketched normal matrix

\[
\tilde{\mathbf{M}} \overset{\text{def}}{=} \tilde{\mathbf{K}}^\top \tilde{\mathbf{K}} + \lambda \mathbf{G}_{(n)}^+ (\mathbf{G}_{(n)}^\top)_+ + \omega \mathbf{N}^\top \mathbf{N},
\]

with probability at least \( 1 - \delta/I_n \), by Lemma 4.3.0.3. Thus, we can use the (non-sketched) matrix \( \mathbf{M}^+ \) as a preconditioner and exploit its Kronecker structure since this iterative method converges in \( \tilde{\mathcal{O}}(1) \) steps by Lemma 4.4.1.1.

It remains to show the main difference with Theorem 4.4.3.3: the time complexity of one Richardson iteration (line 14 of Algorithm 18). We show in Lemma 4.5.0.4 how \( \mathbf{M}^+ \mathbf{x} \) can be computed in time

\[
\tilde{\mathcal{O}} \left( R_{\tilde{x}n}^2 \varepsilon^{-1} \log(I_n/\delta) + R \sum_{k=1}^N R_k + R_\omega \right)
\]

using the Woodbury matrix identity since \( \mathbf{M}^+ \) is a rank-\( R_n \) update to \( (\mathbf{K}^\top \mathbf{K} + \omega \mathbf{I})^+ \).

The solution of each sketch is a \( (1 + \varepsilon) \)-approximation to the optimal factor row by Lemma 4.3.0.4, and the success guarantee follows from a union bound over all \( I_n \) rows. \( \square \)

**Lemma 4.5.0.4.** \textbf{Line 13 in FastFactorMatrixUpdate} takes

\[
\tilde{\mathcal{O}} \left( R_{\tilde{x}n}^2 \varepsilon^{-1} \log(I_n/\delta) + R \sum_{k=1}^N R_k \right)
\]
time after preprocessing.

Proof. Recall that \( \mathbf{N} = \mathbf{I} - \mathbf{G}_n^\top (\mathbf{G}_n^\top)^+ \) and consider the following equality. Letting \( \mathbf{M} \) define

\[
\mathbf{M} = \begin{bmatrix}
\mathbf{K} & \sqrt{\lambda} (\mathbf{G}_n^\top)^+\\
\sqrt{\lambda} (\mathbf{G}_n^\top)^+ & \mathbf{K} \\
\sqrt{\lambda} \mathbf{N} & \sqrt{\lambda} \mathbf{N}
\end{bmatrix},
\]

we have

\[
\mathbf{M} = \mathbf{K}^\top \mathbf{K} + \lambda \mathbf{G}^+_n (\mathbf{G}_n^\top)^+ + w \mathbf{N}^\top \mathbf{N}
\]

For any matrix, we have the pseudoinverse identity \( \mathbf{A}^+ \mathbf{A} \mathbf{A}^\top = \mathbf{A}^\top \), so it follows that

\[
\mathbf{G}^+_n \mathbf{G}_n \mathbf{G}^\top_n (\mathbf{G}_n^\top)^+ = \mathbf{G}_n^\top \mathbf{G}_n^+.
\]

Therefore,

\[
\mathbf{M} = (\mathbf{K}^\top \mathbf{K} + w \mathbf{I}) + \lambda \mathbf{G}^+_n (\mathbf{G}_n^\top)^+ - w (\mathbf{G}^+_n \mathbf{G}_n \mathbf{G}^\top_n (\mathbf{G}_n^\top)^+ - \mathbf{G}^+_n (\mathbf{G}_n^\top)^+)
\]

Applying the Woodbury matrix identity, we have

\[
\mathbf{M}^+ = (\mathbf{K}^\top \mathbf{K} + w \mathbf{I})^{-1}
\]
\[+ (K^	op K + wI)^{-1} G^+_{(n)} (I + (\lambda(G^T_{(n)})^+ - wG_{(n)})(K^	op K + wI)^{-1}G^+_{(n)})^{-1} \]

\[
\cdot (\lambda(G^T_{(n)})^+ - wG_{(n)})(K^	op K + wI)^{-1}.
\]

First note that

\[I + (\lambda(G^T_{(n)})^+ - wG_{(n)})(K^	op K + wI)^{-1}G^+_{(n)} \in \mathbb{R}^{R_n \times R_n}.\]

The time complexity of computing the factored SVD of \(K\) is \(O(\sum_{k \neq n}(I_k R^2_k + R^\omega_k))\). In TuckerALS, these are computed at the end of each factor matrix update and therefore do not need to be computed in this step. After this, multiplying a vector by \((K^	op K + wI)^{-1}\) can be done in time \(O(R_{\neq n} \sum_{m \neq n} R_m)\) by Lemma 4.4.2.1. Therefore, computing

\[I + (\lambda(G^T_{(n)})^+ - wG_{(n)})(K^	op K + wI)^{-1}G^+_{(n)}\]

takes \(O(R \sum_{k=1}^N R_k)\) time. Computing the inverse of this matrix takes \(O(R^\omega_n)\) time. Moreover, this inverse can be used for all Richardson iteration steps and row updates. Finally, observe that multiply any vector with \(G^+_{(n)}\) or \((\lambda(G^T_{(n)})^+ - wG_{(n)})\) takes \(O(R)\) time. Therefore, to evaluate \(M^+z\) for any \(z\), we use (Equation 4.5.6) and repeatedly apply matrix-vector multiplications from right to left. The total running time per evaluation after preprocessing is

\[O \left( R \sum_{k=1}^N R_k \right).\]

Now we show that the vector

\[
\check{K}^	op \check{K}z + \lambda G^+_{(n)} (G^T_{(n)})^+ z + wN^\top Nz - \check{K}^	op b
\]

can be computed fast enough. By the same argument above, this is equivalent to

\[(\check{K}^	op \check{K} + wI)z + \lambda G^+_{(n)} (G^T_{(n)})^+ z - wG^+_{(n)} G_{(n)} z - \check{K}^\top b.\]
Figure 4.1: Running times of Kronecker regression algorithms with a design matrix of size $n^2 \times d^2$.

We can compute $\tilde{K}^\top \tilde{b}$ in $\tilde{O}(R^2 \log(I_n/\delta))$ time, $\lambda G^+_{(n)} (G^T_{(n)})^\top z$ and $w G^+_{(n)} G_{(n)} z$ take $O(R)$ time, and $(\tilde{K}^\top \tilde{K} + w I) z$ takes $O(R + R^2 \log(I_n/\delta))$ time. Summing all of these running times completes the proof.

**Corollary 4.5.0.5.** FastFactorMatrixUpdate updates $A(k) \in \mathbb{R}^{I_k \times R_k}$ in

$$\tilde{O} \left( I_k R^{2-\theta^*} \log(1/\delta) + I_k R \sum_{n=1}^N R_n + R^2 \theta^* \right)$$

time, where $\theta^* > 0$ is the optimally balanced MM exponent in Theorem 4.4.2.4.

### 4.6 Experiments

All experiments were run using NumPy [187] with an Intel Xeon W-2135 processor (8.25MB cache, 3.70 GHz) and 128GB of RAM. The FastKroneckerRegression-based ALS experiments for low-rank Tucker decomposition on image tensors are deferred to Section 4.6.1. All of our code is available online.\(^2\)

**Kronecker regression.** We build on the numerical experiments in [134, 56] for Kronecker regression that use two random factor matrices. We generate matrices $A^{(1)}, A^{(2)} \in \mathbb{R}^{n \times d}$ where each entry is drawn i.i.d. from the normal distribution $N(1, 0.001)$ and compare several algorithms for solving $\min_x \| (A^{(1)} \otimes A^{(2)}) x - 1_n \|_2^2 + \lambda \| x \|_2^2$ as we increase

\(^2\)https://github.com/fahrbach/subquadratic-kronecker-regression
The running times are plotted in Figure 4.1.

The algorithms we compare are: (1) a baseline that solves the normal equation

$$\left( K^T K + \lambda I \right)^+ K^T b$$

and fully exploits the Kronecker structure of $K^T K$ before calling `np.linalg.pinv()`; (2) an enhanced baseline that combines the SVDs of $A^{(n)}$ with Lemma 4.4.2.1, for example, $\text{KronMatMul}([((U^{(1)}))^T, (U^{(2)}))^T], b)$, using only Kronecker-vector products; (3) the sketching algorithm of [56, Algorithm 1]; and (4) our FastKroneckerRegression algorithm in Algorithm 17. For both sketching algorithms, we use $\varepsilon = 0.1$ and $\delta = 0.01$. We reduce the number of row samples in both algorithms by $\alpha = 10^{-5}$ so that the algorithms are more practical and comparable to the earlier experiments in [134, 56]. Lastly, we set $\lambda = 10^{-3}$.

The running times in Figure 4.1 demonstrate several different behaviors. The naive baseline quickly becomes impractical for moderately large values of $n$ or $d$. KronMatMul is competitive for $n \leq 10^4$, especially since it is an exact method. The runtimes of the sketching algorithms are nearly-independent of $n$. [56] works well for small $d$, but deteriorates tremendously as $d$ grows because it computes $((SK)^T SK + \lambda I)^+ \in \mathbb{R}^{d^2 \times d^2}$ and cannot exploit the Kronecker structure of $K$, which takes $O(d^6)$ time.

FastKroneckerRegression, on the other hand, runs in $O(d^4)$ time because it uses quadratic-time Kronecker-vector products in each Richardson iteration step (Algorithm 17).

These experiments also show that combining sketching with iterative methods can give better sketch efficiency. Table 4.2 compares the loss of FastKroneckerRegression and [56, Algorithm 1] to an exact baseline OPT for $d = 64$. Both algorithms use the exact same sketch $SK$ for each value of $n$. Our algorithm uses the original $(K^T K + \lambda I)^+$ as a preconditioner to solve the sketched problem, whereas [56, Algorithm 1] computes
Table 4.2: Kronecker regression losses for \( d = 64 \). OPT denotes the loss of the KronMatMul algorithm, DJSSW19 is [56, Algorithm 1], and Algorithm 17 is FastKroneckerRegression. We also record the relative error of each algorithm and the number of rows sampled from \( A^{(1)} \otimes A^{(2)} \).

<table>
<thead>
<tr>
<th>( n )</th>
<th>OPT</th>
<th>Algorithm 17</th>
<th>Approx</th>
<th>DJSSW19</th>
<th>Approx</th>
<th>Rows sampled (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1024</td>
<td>0.031</td>
<td>0.032</td>
<td>1.051</td>
<td>0.035</td>
<td>1.138</td>
<td>0.0370</td>
</tr>
<tr>
<td>2048</td>
<td>0.123</td>
<td>0.126</td>
<td>1.026</td>
<td>1.577</td>
<td>12.792</td>
<td>0.0093</td>
</tr>
<tr>
<td>4096</td>
<td>0.507</td>
<td>0.520</td>
<td>1.026</td>
<td>275.566</td>
<td>543.776</td>
<td>0.0023</td>
</tr>
<tr>
<td>8192</td>
<td>2.073</td>
<td>2.136</td>
<td>1.030</td>
<td>333.430</td>
<td>160.809</td>
<td>0.0006</td>
</tr>
<tr>
<td>16384</td>
<td>8.238</td>
<td>8.608</td>
<td>1.045</td>
<td>546391.728</td>
<td>66329.791</td>
<td>0.0001</td>
</tr>
</tbody>
</table>

\(((SK)^T SK + \lambda I)^+ (SK)^T Sb\) exactly and becomes numerically unstable for \( n \geq 2048 \) when \( d \in \{16, 32, 64\} \). This raises the question of how to combine sketched information with the original data to achieve more efficient algorithms, even when solving sketched instances. We leave this question of sketch efficiency as an interesting future work.

4.6.1 Low-rank Tucker Decomposition of Image Tensors

Here we compare different Kronecker regression algorithms in the core update of the alternating least squares (ALS) algorithm for Tucker decompositions. For the sketching-based algorithms, we increase the number of row samples to study how this affects the quality of the tensor decomposition. We record the quality of the tensor decomposition using the relative reconstruction error \( \| \hat{X} - X \|_F^2 / \| X \|_F^2 \). The number of row samples used is \( m \in \{1024, 4096, 16384\} \), as in the experiments of [134, 56].

We compare against higher-order orthogonal iteration (HOOI) and ALS as baseline algorithms. We use the Tensorly [188] implementation of HOOI, which is an industry standard. We do not use L2 regularization so that we can compare against HOOI. We compare our Kronecker regression algorithm with [56, Algorithm 1], denoted by DJSSW19. The running times reported are the mean iteration times, where an iteration includes all factor matrix updates and the core tensor update. Trials that ran out of memory or failed to converge are denoted by a dash. All algorithms are run for five iterations.
Table 4.3: Relative reconstruction errors for cardiac MRI tensor with different multilinear ranks.

<table>
<thead>
<tr>
<th>rank</th>
<th>HOOI</th>
<th>ALS</th>
<th>FastKroneckerRegression</th>
<th>DJSSW19</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,1,1,1</td>
<td>0.648</td>
<td>0.648</td>
<td>0.649</td>
<td>0.648</td>
</tr>
<tr>
<td>4,2,2,1</td>
<td>0.569</td>
<td>0.570</td>
<td>0.574</td>
<td>0.571</td>
</tr>
<tr>
<td>4,4,2,2</td>
<td>0.511</td>
<td>0.511</td>
<td>0.533</td>
<td>0.514</td>
</tr>
<tr>
<td>8,2,2,1</td>
<td>0.569</td>
<td>0.577</td>
<td>0.584</td>
<td>0.579</td>
</tr>
<tr>
<td>8,4,4,1</td>
<td>0.448</td>
<td>0.452</td>
<td>0.491</td>
<td>0.459</td>
</tr>
<tr>
<td>8,4,4,2</td>
<td>0.448</td>
<td>0.451</td>
<td>0.492</td>
<td>0.475</td>
</tr>
<tr>
<td>8,8,2,2</td>
<td>0.465</td>
<td>0.467</td>
<td>0.498</td>
<td>0.485</td>
</tr>
<tr>
<td>8,8,4,4</td>
<td>0.350</td>
<td>0.351</td>
<td></td>
<td>0.371</td>
</tr>
</tbody>
</table>

Table 4.4: Average iteration time of ALS with sketching-based Kronecker regression for cardiac MRI tensor with different multilinear ranks (seconds).

<table>
<thead>
<tr>
<th>rank</th>
<th>HOOI</th>
<th>ALS</th>
<th>FastKroneckerRegression</th>
<th>DJSSW19</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,1,1,1</td>
<td>1.187</td>
<td>1.307</td>
<td>1.328</td>
<td>1.334</td>
</tr>
<tr>
<td>4,2,2,1</td>
<td>1.429</td>
<td>1.368</td>
<td>1.345</td>
<td>1.326</td>
</tr>
<tr>
<td>4,4,2,2</td>
<td>1.458</td>
<td>1.463</td>
<td>1.539</td>
<td>1.511</td>
</tr>
<tr>
<td>8,2,2,1</td>
<td>2.401</td>
<td>1.339</td>
<td>1.421</td>
<td>1.415</td>
</tr>
<tr>
<td>8,4,4,1</td>
<td>1.664</td>
<td>1.435</td>
<td>1.575</td>
<td>1.562</td>
</tr>
<tr>
<td>8,4,4,2</td>
<td>1.745</td>
<td>1.614</td>
<td>1.782</td>
<td>1.754</td>
</tr>
<tr>
<td>8,8,2,2</td>
<td>1.741</td>
<td>1.466</td>
<td>1.621</td>
<td>1.810</td>
</tr>
<tr>
<td>8,8,4,4</td>
<td>1.784</td>
<td>1.835</td>
<td>2.131</td>
<td>2.745</td>
</tr>
</tbody>
</table>

Cardiac MRI. This dataset is a 256 × 256 × 14 × 20 tensor whose elements are MRI measurements indexed by \((x, y, z, t)\) where \((x, y, z)\) is a point in space and \(t\) corresponds to time.

We also investigate how sensitive the convergence rate of sketching-based ALS is to the choice of the error parameter \(\varepsilon\). First, we reduce the number of sampled rows by \(\alpha = 0.001\) to compensate for the large constant coefficient in Line 8 in Algorithm 17; otherwise, we do not see any quality degradation even for \(\varepsilon = 0.99\). Then in Table 4.5 and Table 4.6, we compare the RRE at each step of ALS (without sampling) and when using FastKroneckerRegression as a subroutine for decreasing values of \(\varepsilon\).

Hyperspectral. This dataset is a 1024 × 1344 × 33 tensor of time-lapse hyperspectral radiance images capturing a 1-hour interval of a nature scene undergoing illumination.
Table 4.5: Relative reconstruction errors for cardiac MRI tensor with multilinear rank (4, 4, 2, 2) during ALS with and without using FastKroneckerRegression as a subroutine.

<table>
<thead>
<tr>
<th>Step</th>
<th>ALS</th>
<th>ε = 0.8</th>
<th>ε = 0.4</th>
<th>ε = 0.2</th>
<th>ε = 0.1</th>
<th>ε = 0.05</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.55883</td>
<td>0.56270</td>
<td>0.56044</td>
<td>0.55957</td>
<td>0.55942</td>
<td>0.55899</td>
</tr>
<tr>
<td>2</td>
<td>0.51292</td>
<td>0.51609</td>
<td>0.51511</td>
<td>0.51443</td>
<td>0.51377</td>
<td>0.51316</td>
</tr>
<tr>
<td>3</td>
<td>0.51096</td>
<td>0.51466</td>
<td>0.51206</td>
<td>0.51167</td>
<td>0.51139</td>
<td>0.51120</td>
</tr>
<tr>
<td>4</td>
<td>0.51081</td>
<td>0.51338</td>
<td>0.51287</td>
<td>0.51171</td>
<td>0.51127</td>
<td>0.51102</td>
</tr>
<tr>
<td>5</td>
<td>0.51079</td>
<td>0.51361</td>
<td>0.51286</td>
<td>0.51150</td>
<td>0.51126</td>
<td>0.51105</td>
</tr>
</tbody>
</table>

Table 4.6: Relative reconstruction errors for cardiac MRI tensor with multilinear rank (8, 8, 4, 4) during ALS with and without using FastKroneckerRegression as a subroutine.

<table>
<thead>
<tr>
<th>Step</th>
<th>ALS</th>
<th>ε = 0.8</th>
<th>ε = 0.4</th>
<th>ε = 0.2</th>
<th>ε = 0.1</th>
<th>ε = 0.05</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.44961</td>
<td>0.45165</td>
<td>0.45084</td>
<td>0.45021</td>
<td>0.44987</td>
<td>0.44975</td>
</tr>
<tr>
<td>2</td>
<td>0.36573</td>
<td>0.36707</td>
<td>0.36650</td>
<td>0.36612</td>
<td>0.36609</td>
<td>0.36579</td>
</tr>
<tr>
<td>3</td>
<td>0.35488</td>
<td>0.35549</td>
<td>0.35571</td>
<td>0.35508</td>
<td>0.35516</td>
<td>0.35504</td>
</tr>
<tr>
<td>4</td>
<td>0.35162</td>
<td>0.35293</td>
<td>0.35238</td>
<td>0.35201</td>
<td>0.35184</td>
<td>0.35177</td>
</tr>
<tr>
<td>5</td>
<td>0.35081</td>
<td>0.35193</td>
<td>0.35149</td>
<td>0.35124</td>
<td>0.35100</td>
<td>0.35092</td>
</tr>
</tbody>
</table>

Table 4.7: Relative reconstruction errors for hyperspectral tensor with different multilinear ranks.

<table>
<thead>
<tr>
<th>rank</th>
<th>HOOI</th>
<th>ALS</th>
<th>FastKroneckerRegression</th>
<th>DJSSW19</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>1024</td>
<td>4096</td>
</tr>
<tr>
<td>1,1,1</td>
<td>0.271</td>
<td>0.271</td>
<td>0.271</td>
<td>0.271</td>
</tr>
<tr>
<td>2,2,2</td>
<td>0.235</td>
<td>0.235</td>
<td>0.236</td>
<td>0.236</td>
</tr>
<tr>
<td>4,4,4</td>
<td>0.203</td>
<td>0.208</td>
<td>0.213</td>
<td>0.211</td>
</tr>
<tr>
<td>8,8,4</td>
<td>0.169</td>
<td>0.170</td>
<td>0.189</td>
<td>0.176</td>
</tr>
<tr>
<td>8,8,8</td>
<td>0.169</td>
<td>0.169</td>
<td>0.213</td>
<td>0.177</td>
</tr>
<tr>
<td>16,16,4</td>
<td>0.133</td>
<td>0.134</td>
<td>–</td>
<td>0.155</td>
</tr>
</tbody>
</table>

changes [58]. These hyperspectral images and the COIL-100 dataset have both been used recently as benchmark tasks for low-rank tensor decomposition [139, 189, 136].

**COIL-100.** This dataset is a 7200×120×120×3 tensor that contains 7200 colored images of 100 objects (72 images per object). These objects have a wide variety of geometric characteristics and reflective properties. To construct this dataset, these objects were placed on a rotating table and pictures were taken at pose intervals of 5 degrees [140].
Table 4.8: Average iteration time of ALS with sketching-based Kronecker regression for the hyperspectral image tensor with different multilinear ranks (seconds).

<table>
<thead>
<tr>
<th>rank</th>
<th>FastKroneckerRegression</th>
<th>DJSSW19</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>rank</td>
<td>HOOI</td>
</tr>
<tr>
<td>1,1,1</td>
<td>1.873</td>
<td>2.377</td>
</tr>
<tr>
<td>2,2,2</td>
<td>2.019</td>
<td>2.491</td>
</tr>
<tr>
<td>4,4,4</td>
<td>2.255</td>
<td>2.965</td>
</tr>
<tr>
<td>8,8,4</td>
<td>2.845</td>
<td>3.282</td>
</tr>
<tr>
<td>8,8,8</td>
<td>2.888</td>
<td>4.043</td>
</tr>
<tr>
<td>16,16,4</td>
<td>3.997</td>
<td>3.880</td>
</tr>
</tbody>
</table>

Table 4.9: Relative reconstruction errors for the COIL-100 tensor with different multilinear ranks.

<table>
<thead>
<tr>
<th>rank</th>
<th>FastKroneckerRegression</th>
<th>DJSSW19</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>rank</td>
<td>HOOI</td>
</tr>
<tr>
<td>1,1,1,1</td>
<td>0.528</td>
<td>0.528</td>
</tr>
<tr>
<td>4,2,2,1</td>
<td>0.460</td>
<td>0.460</td>
</tr>
<tr>
<td>8,2,2,1</td>
<td>0.460</td>
<td>0.460</td>
</tr>
<tr>
<td>8,4,4,1</td>
<td>0.414</td>
<td>0.414</td>
</tr>
<tr>
<td>8,4,4,2</td>
<td>0.379</td>
<td>0.386</td>
</tr>
<tr>
<td>16,4,4,2</td>
<td>0.349</td>
<td>0.349</td>
</tr>
</tbody>
</table>

Table 4.10: Average iteration time of ALS with sketching-based Kronecker regression for the COIL-100 tensor with different multilinear ranks (seconds).

<table>
<thead>
<tr>
<th>rank</th>
<th>FastKroneckerRegression</th>
<th>DJSSW19</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>rank</td>
<td>HOOI</td>
</tr>
<tr>
<td>1,1,1,1</td>
<td>2.455</td>
<td>10.975</td>
</tr>
<tr>
<td>8,2,2,1</td>
<td>12.092</td>
<td>11.727</td>
</tr>
</tbody>
</table>

4.7 Missing Analysis from section 4.3

Here we show how to use leverage scores of the design matrix $A \in \mathbb{R}^{n \times d}$ to create a smaller least squares problem whose solution vector gives a $(1 + \varepsilon)$-approximation to the original regression problem. Our proof relies on several sketching and leverage score sampling results in randomized numerical linear algebra [177, 178, 175]. These prerequisite results are well-known, but scattered through the literature. They are the building blocks for proving...
our approximate block-regression results in Lemma 4.3.0.4 and Corollary 4.3.0.5.

4.7.1 Approximate Least Squares

We follow the outline of [176] (originally written in [152, Appendix B]). Consider the overdetermined least squares problem defined by a matrix \( A \in \mathbb{R}^{n \times d} \) and response vector \( b \in \mathbb{R}^n \), where \( n \geq d \) and \( \text{rank}(A) = d \). Define the optimal sum of squared residuals to be

\[
R^2 = \min_{x \in \mathbb{R}^d} \|Ax - b\|_2^2. \tag{4.7.1}
\]

Assume for now \( A \) is full rank. Let the compact SVD of the design matrix be \( A = U_A \Sigma_A V_A^T \). By definition, \( U_A \in \mathbb{R}^{n \times d} \) is an orthonormal basis for the column space of \( A \). Let \( U_A^+ \in \mathbb{R}^{n \times (n-d)} \) be an orthonormal basis for the \((n-d)\)-dimensional subspace that is orthogonal to the column space of \( A \). For notational simplicity, let \( b^+ = U_A^+ U_A^T b \) denote the projection of \( b \) onto the orthogonal subspace \( U_A^+ \). The vector \( b^+ \) is important because its norm is equal to the norm of the residual vector. To see this, observe that \( x \) can be chosen so that \( Ax \) perfectly matches the part of \( b \) in the column space of \( A \), but cannot (by definition) match anything in the range of \( U_A^+ \):

\[
R^2 = \min_{x \in \mathbb{R}^d} \|Ax - b\|_2^2 = \|U_A^+ U_A^T b\|_2^2 = \|b^+\|_2^2. \tag{4.7.2}
\]

We denote the solution to the least squares problem by \( x_{opt} \), hence we have \( b = Ax_{opt} + b^+ \).

Now we build on a structural result of [178] that establishes sufficient conditions on any sketching matrix \( S \in \mathbb{R}^{s \times n} \) such that the solution \( \tilde{x}_{opt} \) to the approximate least squares problem

\[
\tilde{x}_{opt} = \arg \min_{x \in \mathbb{R}^d} \|S(Ax - b)\|_2^2 \tag{4.7.3}
\]
gives a relative-error approximation to the original least squares problem. The two condi-
tions we require of matrix $S$ are:

$$\sigma_{\text{min}}^2(S_A) \geq 1/\sqrt{2}, \text{ and}$$

$$\|U_A^TSb^\perp\|_2^2 \leq \varepsilon R^2/2,$$

for some $\varepsilon \in (0, 1)$. While the algorithms we consider in this work are randomized, the following lemma is a deterministic statement. Failure probabilities enter our analysis later when we show our sketch matrices satisfy conditions (Equation 4.7.4) and (Equation 4.7.5) with sufficiently high probability.

**Lemma 4.7.1.1** ([178, Lemma 1]). Consider the overconstrained least squares approximation problem in (Equation 4.7.1), and let the matrix $U_A \in \mathbb{R}^{n \times d}$ contain the top $d$ left singular vectors of $A$. Assume the matrix $S$ satisfies conditions (Equation 4.7.4) and (Equation 4.7.5) for some $\varepsilon \in (0, 1)$. Then, the solution $\tilde{x}_{\text{opt}}$ to the approximate least squares problem (Equation 4.7.3) satisfies:

$$\|Ax_{\text{opt}} - b\|_2^2 \leq (1 + \varepsilon) R^2, \text{ and}$$

$$\|\tilde{x}_{\text{opt}} - x_{\text{opt}}\|_2^2 \leq \frac{1}{\sigma_{\text{min}}^2(A)} \varepsilon R^2.$$

**Proof.** Let us first rewrite the sketched least squares problem induced by $S$ as

$$\min_{x \in \mathbb{R}^d} \|SAx - Sb\|_2^2 = \min_{y \in \mathbb{R}^d} \|SA(x_{\text{opt}} + y) - S(Ax_{\text{opt}} + b^\perp)\|_2^2$$

$$= \min_{y \in \mathbb{R}^d} \|SAy - Sb^\perp\|_2^2$$

$$= \min_{z \in \mathbb{R}^d} \|SU_Az - Sb^\perp\|_2^2.$$  

(Equation 4.7.8) is true because $b = Ax_{\text{opt}} + b^\perp$, and (Equation 4.7.9) follows because the columns of $A$ span the same subspace as the columns of $U_A$. Now, let $z_{\text{opt}} \in \mathbb{R}^d$ be such that $U_Az_{\text{opt}} = A(\tilde{x}_{\text{opt}} - x_{\text{opt}})$ and note that $z_{\text{opt}}$ minimizes (Equation 4.7.9). This fact
follows from

\[ \|SA(\tilde{x}_{opt} - x_{opt}) - Sb^\perp\|_2^2 = \|SA\tilde{x}_{opt} - S(b - b^\perp) - Sb^\perp\|_2^2 = \|SA\tilde{x}_{opt} - Sb\|_2^2. \]

Thus, by the normal equations, we have

\[ (SU_A)^\top SU_A z_{opt} = (SU_A)^\top Sb^\perp. \]

Taking the norm of both sides and observing that under condition (Equation 4.7.4) we have

\[ \sigma_i((SU_A)^\top SU_A) = \sigma_i^2(SU_A) \geq 1/\sqrt{2}, \text{ for all } i \in [d], \]

it follows that

\[ \|z_{opt}\|_2^2/2 \leq \|(SU_A)^\top SU_A z_{opt}\|_2 = \|(SU_A)^\top Sb^\perp\|_2. \]  \hspace{1cm} (4.7.10)

Using condition (Equation 4.7.5), we observe that

\[ \|z_{opt}\|_2^2 \leq 2\|(SU_A)^\top Sb^\perp\|_2 \leq \epsilon R^2. \]  \hspace{1cm} (4.7.11)

To establish the first claim of the lemma, let us rewrite the squared norm of the residual vector as

\[ \|Ax_{opt} - b\|_2^2 = \|Ax_{opt} - Ax_{opt} + Ax_{opt} - b\|_2^2 \]
\[ = \|Ax_{opt} - Ax_{opt}\|_2^2 + \|Ax_{opt} - b\|_2^2 \]  \hspace{1cm} (4.7.12)
\[ = \|U_A z_{opt}\|_2^2 + R^2 \]  \hspace{1cm} (4.7.13)
\[ \leq (1 + \epsilon) R^2, \]  \hspace{1cm} (4.7.14)

where (Equation 4.7.12) follows from the Pythagorean theorem since \( b - Ax_{opt} = b^\perp \), which is orthogonal to \( A \), and consequently \( A(\tilde{x}_{opt} - x_{opt}) \); (Equation 4.7.13) follows from the definition of \( z_{opt} \) and \( R^2 \); and (Equation 4.7.14) follows from (Equation 4.7.11)
and the orthogonality of $U_A$.

To establish the second claim of the lemma, recall that $A(x_{\text{opt}} - \tilde{x}_{\text{opt}}) = U_A z_{\text{opt}}$. Taking the norm of both sides of this expression, we have

$$\|x_{\text{opt}} - \tilde{x}_{\text{opt}}\|_2^2 \leq \frac{\|U_A z_{\text{opt}}\|_2^2}{\sigma_{\text{min}}^2(A)}$$

(4.7.15)

$$\leq \frac{\varepsilon R^2}{\sigma_{\text{min}}^2(A)},$$

(4.7.16)

where (Equation 4.7.15) follows since $\sigma_{\text{min}}(A)$ is the smallest singular value of $A$ and $\text{rank}(A) = d$; and (Equation 4.7.16) follows from (Equation 4.7.11) and the orthogonality of $U_A$. □

Next we present two results that are useful for proving our sketches $S$ satisfy the structural conditions in (Equation 4.7.4) and (Equation 4.7.5). The first result states $SU_A$ is a subspace embedding for the column space of $U_A$. This result can be thought of as an approximate isometry and is noticeably stronger than the desired condition $\sigma_{\text{min}}^2(SU_A) \geq 1/\sqrt{2}$.

**Theorem 4.7.1.2** ([175, Theorem 17]). Consider $A \in \mathbb{R}^{n \times d}$ and its compact SVD $A = U_A \Sigma_A V_A^T$. Let $p \in [0, 1]^n$ be a $\beta$-overestimate for the leverage score distribution of $A$. Let $s > 144d \ln(2d/\delta)/{(\beta \varepsilon^2)}$. Let the matrix $S \in \mathbb{R}^{s \times n}$ be the output of $\text{SampleRows}(A, s, p)$ (Definition 4.3.0.2). Then, with probability at least $1 - \delta$, simultaneously for all $i$, we have

$$1 - \varepsilon \leq \sigma_i^2(SU_A) \leq 1 + \varepsilon.$$

For the second structural condition, we use the following result about squared-distance sampling for approximate matrix multiplication in [177]. In our analysis of block leverage score sampling (e.g., ridge regression), it is possible (and beneficial) that $\beta > 1$ and that rows are sometimes not sampled. We modify the original theorem statement and provide a proof to show that the result is unaffected.
Theorem 4.7.1.3 ([177, Lemma 8]). Let \( A \in \mathbb{R}^{n \times m} \), \( B \in \mathbb{R}^{n \times p} \), and \( s \) denote the number of samples. Let the vector \( p \in [0, 1]^n \) contain probabilities such that, for all \( i \in [n] \), we have

\[
p_i \geq \beta \frac{\|a_i\|_2^2}{\|A\|_F^2},
\]

for some constant \( \beta > 0 \). We require that \( \|p\|_1 \leq 1 \), but it is possible that \( p \) does not contain all of the probability mass (i.e., \( \|p\|_1 < 1 \)). Sample \( s \) row indices \((\xi^{(1)}, \xi^{(2)}, \ldots, \xi^{(s)})\) from \( p \), independently and with replacement, and form the approximate product

\[
 \frac{1}{s} \sum_{t=1}^s \frac{1}{p_{\xi^{(t)}}} a_{\xi^{(t)};}^\top b_{\xi^{(t)}} = (SA)^\top SB,
\]

where \( S \in \mathbb{R}^{s \times n} \) is the sampling and rescaling matrix whose \( t \)-th row is defined by the entries

\[
s_{tk} = \begin{cases} 
\frac{1}{\sqrt{s} p_k} & \text{if } k = \xi_t, \\
0 & \text{otherwise.}
\end{cases}
\]

Disregard trials that occur with the remaining probability \( 1 - \|p\|_1 \). Then, we have

\[
\mathbb{E} \left[ \|A^\top B - (SA)^\top SB\|_F^2 \right] \leq \frac{1}{\beta s} \|A\|_F^2 \|B\|_F^2.
\]

Proof. First we analyze the entry of \((SA)^\top SB\) at index \((i, j)\). Viewing the approximate product as a sum of outer products, we can write this entry in terms of scalar random variables \( X_t \), for \( t \in [s] \), as follows:

\[
X_t = \begin{cases} 
\frac{a_{\xi^{(t)};} b_{\xi^{(t)}}}{s p_{\xi^{(t)}}} & \text{with probability } p_i \text{ for each } i \in [n], \\
0 & \text{otherwise with probability } 1 - \|p\|_1
\end{cases} \implies \left[(SA)^\top SB\right]_{ij} = \sum_{t=1}^s X_t.
\]
The expected values of $X_t$ and $X_t^2$ for all values of $t$ are

\[
\mathbb{E}[X_t] = \sum_{k=1}^{n} p_k \frac{a_{ki}b_{kj}}{sp_k} = \frac{1}{s}(A^\top B)_{ij}, \quad \text{and} \quad \\
\mathbb{E}[X_t^2] = \sum_{k=1}^{n} p_k \left( \frac{a_{ki}b_{kj}}{sp_k} \right)^2 = \frac{1}{s^2} \sum_{k=1}^{n} \frac{(a_{ki}b_{kj})^2}{p_k}.
\]

Therefore, $\mathbb{E}[(SA)^\top SB]_{ij} = \sum_{t=1}^{s} \mathbb{E}[X_t] = (A^\top B)_{ij}$, which means the estimator is unbiased. Furthermore, since the estimated matrix entry is the sum of $s$ i.i.d. random variables, its variance is

\[
\text{Var} \left( [(SA)^\top SB]_{ij} \right) = \sum_{t=1}^{s} \text{Var} \left( X_t \right) \\
= \sum_{t=1}^{s} \left( \mathbb{E}[X_t^2] - \mathbb{E}[X_t]^2 \right) \\
= \sum_{t=1}^{s} \frac{1}{s^2} \sum_{k=1}^{n} \left( \frac{(a_{ki}b_{kj})^2}{p_k} - (A^\top B)_{ij}^2 \right) \\
= \frac{1}{s} \sum_{k=1}^{n} \left( \frac{(a_{ki}b_{kj})^2}{p_k} - (A^\top B)_{ij}^2 \right).
\]

Now we apply this result to the expectation we want to bound:

\[
\mathbb{E} \left[ \|A^\top B - (SA)^\top SB\|_F^2 \right] = \sum_{i=1}^{m} \sum_{j=1}^{p} \mathbb{E} \left[ \left( [(SA)^\top SB]_{ij} - (A^\top B)_{ij} \right)^2 \right] \\
= \sum_{i=1}^{m} \sum_{j=1}^{p} \mathbb{E} \left[ \left( [(SA)^\top SB]_{ij} - \mathbb{E} \left[ [(SA)^\top SB]_{ij} \right] \right)^2 \right] \\
= \sum_{i=1}^{m} \sum_{j=1}^{p} \text{Var} \left( [(SA)^\top SB]_{ij} \right) \\
= \frac{1}{s} \sum_{i=1}^{m} \sum_{j=1}^{p} \sum_{k=1}^{n} \left( \frac{(a_{ki}b_{kj})^2}{p_k} - (A^\top B)_{ij}^2 \right) \\
= \frac{1}{s} \sum_{k=1}^{n} \left( \sum_{i=1}^{m} a_{ki}^2 \right) \left( \sum_{j=1}^{p} b_{kj}^2 \right) - \frac{n}{s} \sum_{i=1}^{m} \sum_{j=1}^{p} (A^\top B)_{ij}^2.
\]
\[
\begin{align*}
&= \frac{1}{s} \sum_{k=1}^{n} \frac{\|a_k\|_2^2 \|b_k\|_2^2}{p_k} - \frac{n}{s} \|A^T B\|_F^2 \\
&\leq \frac{1}{s} \sum_{k=1}^{n} \frac{\|a_k\|_2^2 \|b_k\|_2^2}{p_k}.
\end{align*}
\]

The last inequality uses the fact that the Frobenius norm of any matrix is nonnegative.

Finally, by using the \(\beta\)-overestimate assumption on the sampling probabilities, we have

\[
\mathbb{E} \left[ \|A^T B - (SA)^T SB\|_F^2 \right] \leq \frac{1}{s} \sum_{k=1}^{n} \frac{\|a_k\|_2^2 \|b_k\|_2^2}{p_k}
\]

\[
\leq \frac{1}{s} \sum_{k=1}^{n} \left( \frac{\|A\|_F^2}{\beta} \frac{\|a_k\|_2^2 \|b_k\|_2^2}{\|a_k\|_2^2} \right)
\]

\[
= \frac{1}{s\beta} \|A\|_F^2 \sum_{k=1}^{n} \|b_k\|_2^2
\]

\[
= \frac{1}{s\beta} \|A\|_F^2 \|B\|_F^2,
\]

which is the desired upper bound. \(\square\)

### 4.7.2 Generalizing to Submatrix Sketching

Now that our main tools are in place, we extend the analysis of approximate least squares to work with sketched submatrices of the vertically stacked block design matrix.

**Lemma 4.3.0.3.** Let \(A = \begin{bmatrix} A_1; A_2 \end{bmatrix}\) be vertically stacked with \(A_1 \in \mathbb{R}^{n_1 \times d}\) and \(A_2 \in \mathbb{R}^{n_2 \times d}\). Let \(p \in [0, 1]^{n_1}\) be a \(\beta\)-overestimate for the leverage score distribution of \(A_1\). If \(s > 144d \ln(2d/\delta)/(\beta\varepsilon^2)\), the sketch \(S\) returned by \text{SampleRows}(A_1, s, p) guarantees, with probability at least \(1 - \delta\), that

\[
(1 - \varepsilon)A^T A \preceq (SA_1)^T S A_1 + A_2^T A_2 \preceq (1 + \varepsilon)A^T A.
\]

**Proof.** Write the compact SVD of \(A_1\) as \(A_1 = U_{A_1} \Sigma_{A_1} V_{A_1}^T\). Theorem 4.7.1.2 guarantees
that with probability at least $1 - \delta$,

$$1 - \varepsilon \leq \sigma^2_f (SU_{A_1}) \leq 1 + \varepsilon.$$ 

Therefore, we have

$$(1 - \varepsilon)I_d \preceq (SU_{A_1})^T SU_{A_1} \preceq (1 + \varepsilon)I_d.$$ 

It follows that

$$(1 - \varepsilon)A_1^T A_1 = (1 - \varepsilon)V_{A_1} \Sigma_{A_1}^T I_d \Sigma_{A_1} V_{A_1}^T$$

$$\preceq V_{A_1} \Sigma_{A_1}^T U_{A_1}^T S^T SU_{A_1} \Sigma_{A_1} V_{A_1}^T$$

$$= (SA_1)^T SA_1.$$ 

Similarly, we have $(SA_1)^T SA_1 \preceq (1 + \varepsilon)A_1^T A_1$. Writing $A^T A = A_1^T A_1 + A_2^T A_2$ as the sum of outer products, we have

$$(1 - \varepsilon) A^T A \preceq (1 - \varepsilon) A_1^T A_1 + A_2^T A_2$$

$$\preceq (SA_1)^T SA_1 + A_2^T A_2$$

$$\preceq (1 + \varepsilon) A_1^T A_1 + A_2^T A_2$$

$$\preceq (1 + \varepsilon) A^T A,$$

which completes the proof. \qed

**Lemma 4.3.0.4** (Approximate block regression). Consider the problem

$$\arg \min_{x \in \mathbb{R}^d} \|Ax - b\|_2^2$$

where $A = [A_1; A_2]$ and $b = [b_1; b_2]$ are vertically stacked and $A_1 \in \mathbb{R}^{n_1 \times d}$, $A_2 \in \mathbb{R}^{n_2 \times d}$, $b_1 \in \mathbb{R}^{n_1}$, $b_2 \in \mathbb{R}^{n_2}$. Let $p \in [0, 1]^{n_1}$ be a $\beta$-overestimate for the leverage score distribution.
of $A_1$. Let $s \geq 1680d \ln(40d)/(\beta \varepsilon)$ and let $S$ be the output of $\text{SampleRows}(A_1, s, p)$. If

$$\tilde{x}^* = \arg \min_{x \in \mathbb{R}^d} \left( \|S(A_1 x - b_1)\|_2^2 + \|A_2 x - b_2\|_2^2 \right),$$

then, with probability at least $9/10$, we have

$$\|A\tilde{x}^* - b\|_2^2 \leq (1 + \varepsilon) \min_{x \in \mathbb{R}^d} \|Ax - b\|_2^2.$$ 

Proof. Let $\delta = 1/10$ be the desired failure probability. Consider the augmented sketch matrix

$$S' = \begin{bmatrix} S & 0_{s \times n_2} \\ 0_{n_2 \times n_1} & I_{n_2} \end{bmatrix}. \quad (4.7.17)$$

It follows that

$$S'A = \begin{bmatrix} SA_1 \\ A_2 \end{bmatrix}. \quad (4.7.18)$$

Let the compact SVD of $A$ be $A = U\Sigma V^\top$. We prove that each of the structural conditions about $S'$ in (Equation 4.7.4) and (Equation 4.7.5) fail with probability at most $\delta/2$. Then we use a union bound and apply Lemma 4.7.1.1.

Satisfying structural condition 1. It follows from (Equation 4.7.18) that

$$(S'A)^\top S'A = (SA_1)^\top SA_1 + A_2^\top A_2.$$ 

Using Lemma 4.3.0.3, we know

$$(1 - \varepsilon) A^\top A \preceq (S'A)^\top S'A \preceq (1 + \varepsilon) A^\top A. \quad (4.7.19)$$
Since \( A^T A = V \Sigma^T I d \Sigma V^T \) and \((S'A^T) S'A = V \Sigma^T U^T S'U \Sigma V^T \), it follows from (Equation 4.7.19) that
\[
(1 - \varepsilon) I_d \preceq (S'U)^T S'U \preceq (1 + \varepsilon) I_d
\]
since \( \Sigma \) and \( V^T \) are positive definite. Thus, the first structural condition (Equation 4.7.4) is true with probability at least \( 1 - \delta/2 \) as long as \( 1 - \varepsilon \geq 1/\sqrt{2} \). This means the number of samples needs to be at least
\[
s > \frac{144d \ln(4d/\delta)}{\beta(1 - 1/\sqrt{2})^2} > \frac{1680d \ln(4d/\delta)}{\beta}.
\]

**Satisfying structural condition 2.** We show (Equation 4.7.5) holds with probability at least \( 1 - \delta/2 \) using a modification of Theorem 4.7.1.3 and Markov’s inequality. First observe that
\[
U^T b^\perp = U^T \left(U^\perp U^\perp^T b\right) = 0_{\text{rank}(A)},
\]
where \( b^\perp \) is defined as in Section 4.7.1. Thus, the second structural condition can be seen as bounding how closely this sampled product approximates the zero vector. It follows that
\[
\left\| U^T S'S'b^\perp \right\|^2_2 = \left\| U^T b^\perp - U^T S'S'b^\perp \right\|^2_2
\]
\[
= \left\| U^T \left(I_{n_1+n_2} - S^TS'\right) b^\perp \right\|^2_2
\]
\[
= \left\| U^T \begin{bmatrix} I_{n_1} - S^TS & 0 \\ 0 & 0 \end{bmatrix} b^\perp \right\|^2_2
\]
\[
= \left\| \tilde{U}^T \left(I_{n_1} - S^TS\right) \tilde{b}^\perp \right\|^2_2
\]
\[
= \left\| \tilde{U}^T b^\perp - \tilde{U}^T S^T \tilde{b}^\perp \right\|^2_2,
\]
where \( \tilde{U} \in \mathbb{R}^{n_1 \times d} \) and \( \tilde{b}^\perp \in \mathbb{R}^{n_1} \) denote the first \( n_1 \) rows of \( U \) and \( b^\perp \), respectively.

Now we bound the probability that a row index in \( \tilde{U} \) is sampled when constructing \( S \),
which allows us to apply Theorem 4.7.1.3:

\[
\Pr \left( \text{row } i \in [n_1] \text{ is sampled} \right) \geq \beta \frac{\ell_i(A_1)}{\|\ell(A_1)\|} \tag{4.7.20}
\]

\[
= \beta \frac{\ell_i(A_1)}{\text{rank}(A_1)} \cdot \frac{\|\ell_i(A)\|_1}{\ell_i(A)} \cdot \frac{\ell_i(A)}{\|\ell_i(A)\|_1} \tag{4.7.21}
\]

\[
\geq \beta \frac{\|\ell_i(A)\|_1}{\text{rank}(A_1)} \cdot \frac{\ell_i(A)}{\|\ell_i(A)\|_1} \tag{4.7.22}
\]

\[
= \beta \frac{\|\tilde{U}\|_F^2}{\text{rank}(A_1)} \cdot \frac{\|\tilde{u}_i\|_2^2}{\|\tilde{U}\|_F^2}. \tag{4.7.23}
\]

We use \(\|\ell_i(A)\|_1 = \sum_{i \in [n_1]} \ell_i(A)\) to denote the sum of leverage scores of \(A\) corresponding to the rows of \(A_1\). (Equation 4.7.22) holds because leverage scores do not increase when rows are added to the matrix, i.e., \(\ell_i(A_1) \geq \ell_i(A)\). (Equation 4.7.23) is true because the leverage scores of \(A\) corresponding to the rows in \(A_1\) are given by the submatrix \(\tilde{U}\) in the compact SVD of \(A\). Therefore, Theorem 4.7.1.3 guarantees that

\[
\|U^T S'^T S' b^\perp\|_2^2 = \|\tilde{U}^T \tilde{b}^\perp - (SU)^T S \tilde{b}^\perp\|_2^2 \\
\leq \frac{\text{rank}(A_1)}{\beta \|\tilde{U}\|_F^2} \cdot \|\tilde{U}\|_F^2 \cdot \|\tilde{b}^\perp\|_2^2 \\
\leq \frac{\text{rank}(A_1)}{\beta s} \cdot \|b^\perp\|_2^2.
\]

Since \(b^\perp\) is the residual vector, applying Markov’s inequality gives us

\[
\Pr \left( \|U^T S'^T S' b^\perp\|_2^2 \geq \epsilon \frac{\|b^\perp\|_2^2}{2} \right) \leq \frac{\text{rank}(A_1)}{\beta s} \cdot \|b^\perp\|_2^2 \cdot \frac{2}{\epsilon \|b^\perp\|_2^2} = \frac{2 \cdot \text{rank}(A_1)}{\beta s \epsilon}. \tag{4.7.24}
\]

To upper bound (Equation 4.7.24) by a failure probability of \(\delta/2\), the number of samples needs to be at least

\[
s \geq \frac{4 \cdot \text{rank}(A_1)}{\beta \delta \epsilon}.
\]
Conclusion. Since \( d \geq \text{rank}(A_1) \) and \( \delta = 1/10 \), it follows that

\[
\max\left\{ \frac{1680d \ln(4d/\delta)}{\beta}, \frac{4d}{\beta \delta \epsilon} \right\} \leq \frac{1680d \ln(40d)}{\beta \epsilon} \leq s
\]

samples are sufficient for both structural conditions to hold at the same time with probability at least \( 1 - (\delta/2 + \delta/2) = 1 - \delta \) by a union bound. Finally, we may apply Lemma 4.7.1.1 to achieve the \((1 + \epsilon)\)-approximation guarantee. \( \Box \)

**Corollary 4.3.0.5.** For any \( A \in \mathbb{R}^{n \times d}, \ b \in \mathbb{R}^d, \lambda \geq 0 \), consider

\[
\arg\min_{x \in \mathbb{R}^d} \left( \|Ax - b\|_2^2 + \lambda \|x\|_2^2 \right).
\]

Let \( s \geq 1680d \ln(40d)/(\beta \epsilon) \) and \( p \in [0, 1]^{n_1} \) be a \( \beta \)-overestimate for the leverage scores of \( A \). If \( S \) is the output of \text{SampleRows}(A, s, p), then, with probability at least \( 9/10 \), the sketched solution

\[
\tilde{x}^* = \arg\min_{x \in \mathbb{R}^d} \left( \|S(Ax - b)\|_2^2 + \lambda \|x\|_2^2 \right)
\]

gives a \((1 + \epsilon)\)-approximation to the original problem.

**Proof.** This is an immediate consequence of our results for approximate block regression in Lemma 4.3.0.4. Consider the augmented matrices

\[
A' = \begin{bmatrix} A \\ \sqrt{\lambda}I_d \end{bmatrix} \quad \text{and} \quad b' = \begin{bmatrix} b \\ 0_d \end{bmatrix}.
\]

For any \( x \in \mathbb{R}^d \), we have

\[
\|A'x - b'\|_2^2 = \left\| \begin{bmatrix} Ax - b \\ \sqrt{\lambda}x \end{bmatrix} \right\|_2^2 = \|Ax - b\|_2^2 + \lambda \|x\|_2^2.
\]
Therefore, it suffices to approximately solve

$$\arg \min_{x \in \mathbb{R}^d} \| A' x - b' \|,$$

so we can use Lemma 4.3.0.4 to complete the proof. \hfill \Box
In this chapter, we study the combinatorial optimization problem of finding an optimal core tensor shape, also called multilinear rank, for a size-constrained Tucker decomposition. We give an algorithm with provable approximation guarantees for its reconstruction error via connections to higher-order singular values. Specifically, we introduce a novel Tucker packing problem, which we prove is NP-hard, and give a polynomial-time approximation scheme based on a reduction to the 2-dimensional knapsack problem with a matroid constraint. We also generalize our techniques to tree tensor network decompositions. We implement our algorithm using an integer programming solver, and show that its solution quality is competitive with (and sometimes better than) the greedy algorithm that uses the true Tucker decomposition loss at each step, while also running up to 1000x faster.

5.1 Introduction

Low-rank tensor decomposition is a powerful tool in the modern machine learning toolbox. Like low-rank matrix factorization, it has countless applications in scientific computing, data mining, and signal processing [129, 131], e.g., anomaly detection in data streams [132] and compressing convolutional neural networks on mobile devices for faster inference while reducing power consumption [59].

The most widely used tensor decompositions are the canonical polyadic (CP) decomposition, Tucker decomposition, and tensor-train decomposition [190]—the last two being instances of tree tensor networks [191]. CP decomposition factors a tensor into the sum of \( r \) rank-one tensors. Tucker decomposition, however, specifies the rank \( R_n \) in each dimension \( n \) and relies on a core tensor \( \mathcal{G} \in \mathbb{R}^{R_1 \times \cdots \times R_N} \) for reconstructing the decomposition.
Figure 5.1: Pareto frontier of core shapes \( r \in [20]^3 \) for hyperspectral tensor \( \mathcal{X} \in \mathbb{R}^{1024 \times 1344 \times 33} \). Plots the RRE, i.e., \( L(\mathcal{X}, r) / \| \mathcal{X} \|_F \), as a function of compression rate. RRE-greedy builds core shapes by computing Tucker decompositions at each step. HOSVD-IP is Algorithm 20 with integer programming, which builds core shapes via a surrogate packing problem on higher-order singular values.

The notion of multilinear rank \( r = (R_1, \ldots, R_N) \) puts practitioners in a challenging spot because the set of feasible core shapes can be exponentially large. Furthermore, searching in this state space can be prohibitively expensive because evaluating the true quality of a core shape requires computing a Tucker decomposition, which for large tensors can take hours and consume hundreds of GB of RAM. For example, in the MATLAB Tensor Toolbox [133], we need to specify the core shape parameter ranks in advance before computing a size-constrained Tucker decomposition.

In practice, the most popular Tucker decomposition algorithms are the \( r \)-truncated higher-order singular value decomposition (HOSVD) in [55], sequentially truncated ST-HOSVD in [192], and higher-order orthogonal iteration (HOOI), which is a structured alternating least squares algorithm.

We explore the simple but fundamental discrete optimization problem for low-rank tensor decompositions:
If a Tucker decomposition of $\mathcal{X}$ can use at most $c$ parameters, which core tensor shape minimizes the reconstruction error?

This is a multilinear generalization of the best rank-$r$ matrix approximation problem. While there are many parallels to low-rank matrix factorization, tensor rank-related problems can be thoroughly different and more challenging than their matrix counterparts. For example, computing the CP rank of a real-valued tensor is NP-hard [60].

5.1.1 Our contributions and techniques

We summarize the main contributions of this work below:

1. We formalize the core tensor shape problem for size-constrained Tucker decompositions and introduce the Tucker packing problem, which we prove is NP-hard. The approximation algorithms we develop build on a relationship between the optimal reconstruction error of a rank-$r$ Tucker decomposition and a multi-dimensional tail sum of its higher-order singular values [55, 193].

2. We design a polynomial-time approximation scheme (PTAS) for the surrogate Tucker packing problem (Theorem 5.4.2.5) by showing that it suffices to consider a small number of budget splits between the cost of the core tensor and the cost of the factor matrices. Each budget split subproblem reduces to a 2-dimensional knapsack problem with a partition matroid constraint after minor transformations. We solve these knapsack problems using the PTAS of [194], or in practice with integer linear programming.

3. We extend our approach to tree tensor networks, which generalize the Tucker decomposition, tensor-train decomposition, and hierarchical Tucker decomposition. In doing so, we synthesize several works on tree tensors from the mathematics and physics communities, and give a succinct introduction for computer scientists.
4. Finally, we demonstrate the effectiveness of our Tucker packing-based core shape solvers on four real-world tensors. Our HOSVD-IP algorithm is competitive with (and sometimes outperforms) the greedy algorithm that uses the true RRE, while running up to 1000x faster.

5.1.2 Related works

Core shape constraints. [195] introduced the problem of computing the best rank-\(r\) tensor approximation for a prespecified core shape \(r\), and demonstrated the benefit of initializing the decomposition with a truncated HOSVD and then running iterative methods such as HOOI. [196, 197, 198, 199, 200] consider this problem for rank-\((r_1, r_2, r_3)\) decompositions and develop a suite of advanced algorithms: a Newton method on Grassmannian manifolds, a trust-region method on Riemannian manifolds, Jacobi rotations for symmetric tensors, and a Krylov-type iterative method. All these works, however, are concerned with optimizing the tensor decomposition for a fixed core shape—not with optimizing the core tensor shape itself.

[173] and [174] recently explored rank-adaptive methods for HOOI that find minimal core shapes such that the Tucker decomposition achieves a target reconstruct error. They also leverage properties of the HOSVD, but they do not impose a hard constraint on the size of the returned Tucker decomposition. [61] generalized the RRE-greedy algorithm in Figure 5.1 to tensor networks for both rank and size constraints.

Low-rank tensor decomposition. [153] gave polynomial-time \((1 + \varepsilon)\)-approximation algorithms for many types of low-rank tensor decompositions with respect to the Frobenius norm, including CP and Tucker decompositions. [154] showed that if a third-order tensor has an exact Tucker decomposition, then all local minima of an appropriately regularized loss landscape are globally optimal. Several works recently studied Tucker decomposition in streaming models [156, 158] and a sliding window model [132]. Fast randomized low-
rank tensor decomposition algorithms based on sketching have been proposed in [136, 151, 189, 155, 157, 139, 201, 202].

5.2 Preliminaries

Notation. The order of a tensor is its number of dimensions. We denote scalars by normal lowercase letters \( x \in \mathbb{R} \), vectors by boldface lower letters \( \mathbf{x} \in \mathbb{R}^n \), matrices by boldface uppercase letters \( \mathbf{X} \in \mathbb{R}^{m \times n} \), and higher-order tensors by boldface script letters \( \mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_N} \). We use normal uppercase letters for the size of an index set, e.g., \([N] = \{1, 2, \ldots, N\}\). We denote the \( i \)-th entry of vector \( \mathbf{x} \) by \( x_i \), the \((i, j)\)-th entry of matrix \( \mathbf{X} \) by \( x_{ij} \), and the \((i, j, k)\)-th entry of a third-order tensor \( \mathcal{X} \) by \( x_{ijk} \).

Tensor products. The fibers of a tensor are the vectors we get by fixing all but one index. For example, if \( \mathcal{X} \in \mathbb{R}^3 \), we denote the column, row, and tube fibers by \( \mathbf{x}_{j k}, \mathbf{x}_{i k}, \) and \( \mathbf{x}_{ij} \), respectively. The mode-\( n \) unfolding of a tensor \( \mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_N} \) is the matrix \( \mathbf{X}(n) \in \mathbb{R}^{I_n \times (I_1 \cdots I_{n-1}I_{n+1} \cdots I_N)} \) that arranges the mode-\( n \) fibers of \( \mathcal{X} \) as columns of \( \mathbf{X}(n) \) ordered lexicographically by index.

We denote the \( n \)-mode product of a tensor \( \mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_N} \) and matrix \( \mathbf{A} \in \mathbb{R}^{J \times I_N} \) by \( \mathcal{Y} = \mathcal{X} \times_n \mathbf{A} \), where \( \mathcal{Y} \in \mathbb{R}^{I_1 \times \cdots \times I_{n-1}I_{n+1} \cdots I_N \times J} \). This operation multiplies each mode-\( n \) fiber of \( \mathcal{X} \) by \( \mathbf{A} \), and can be expressed element-wise as

\[
(\mathcal{X} \times_n \mathbf{A})_{i_1 \cdots i_{n-1} j n_{n+1} \cdots i_N} = \sum_{i_n=1}^{I_n} x_{i_1 i_2 \cdots i_{n-1} i_n} a_{j i_n},
\]

The inner product of two tensors \( \mathcal{X}, \mathcal{Y} \in \mathbb{R}^{I_1 \times \cdots \times I_N} \) is the sum of the products of their entries:

\[
\langle \mathcal{X}, \mathcal{Y} \rangle = \sum_{i_1=1}^{I_1} \sum_{i_2=1}^{I_2} \cdots \sum_{i_N=1}^{I_N} x_{i_1 i_2 \cdots i_N} y_{i_1 i_2 \cdots i_N}.
\]

The Frobenius norm of a tensor \( \mathcal{X} \) is \( \|\mathcal{X}\|_F = \sqrt{\langle \mathcal{X}, \mathcal{X} \rangle} \).

252
**Tucker decomposition.** The Tucker decomposition of a tensor \( X \in \mathbb{R}^{I_1 \times \cdots \times I_N} \) decomposes \( X \) into a core tensor \( G \in \mathbb{R}^{R_1 \times \cdots \times R_N} \) and \( N \) factor matrices \( A^{(n)} \in \mathbb{R}^{I_n \times R_n} \). We refer to \( r = (R_1, \ldots, R_N) \) as the core shape, which is also called the multilinear rank or truncation of the decomposition. We denote the loss of an optimal rank-\( r \) Tucker decomposition by
\[
L(X, r) \overset{\text{def}}{=} \min_{G \in \mathbb{R}^{R_1 \times \cdots \times R_N}} \left\| X - G \times_1 A^{(1)} \times_2 \cdots \times_N A^{(N)} \right\|_F^2.
\]

### 5.3 Reduction to HOSVD Tucker packing

#### 5.3.1 Higher-order singular value decomposition

We start with a recap of the seminal work on higher-order singular value decompositions (HOSVD) by [55].

**Theorem 5.3.1.1** ([55, Theorem 2]). Any tensor \( X \in \mathbb{R}^{I_1 \times \cdots \times I_N} \) can be written as
\[
X = S \times_1 U^{(1)} \times_2 \cdots \times_N U^{(N)},
\]
where each \( U^{(n)} \in \mathbb{R}^{I_n \times I_n} \) is an orthogonal matrix and \( S \in \mathbb{R}^{I_1 \times \cdots \times I_N} \) is a tensor with subtensors \( S_{i_n=\alpha} \), obtained by fixing the \( n \)-th index to \( \alpha \), that have the properties:

1. all-orthogonality: for all possible values of \( n, \alpha \) and \( \beta \) subject to \( \alpha \neq \beta \), two subtensors \( S_{i_n=\alpha} \) and \( S_{i_n=\beta} \) are orthogonal, i.e., \( (S_{i_n=\alpha}, S_{i_n=\beta}) = 0 \) when \( \alpha \neq \beta \);

2. ordering: for all values of \( n \), \( \| S_{i_n=1} \|_F \geq \| S_{i_n=2} \|_F \geq \cdots \geq \| S_{i_n=I_n} \|_F \geq 0 \).

Furthermore, the values \( \| S_{i_n=\alpha} \|_F \), denoted by \( \sigma^{(n)}_i \), are the singular values of the mode-\( n \) unfolding \( X_{(n)} \), and the columns of \( U^{(n)} \) are the left singular vectors.

Next, we present the TuckerHOSVD algorithm. This is a widely used initialization strategy when computing rank-\( r \) Tucker decompositions [129], i.e., if the core shape \( r \) is predetermined.
Algorithm 19: \texttt{TuckerHOSVD}

1. **Input:** $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$, core shape $r = (R_1, \ldots, R_N)$
2. for $n = 1$ to $N$ do
3. \hspace{0.5cm} $A^{(n)} \leftarrow R_n$ top left singular vectors of $X^{(n)}$
4. \hspace{0.5cm} $\mathcal{G} \leftarrow \mathcal{X} \times_1 A^{(1)} \times_2 \cdots \times_N A^{(N)}$
5. return $\mathcal{G}, A^{(1)}, A^{(2)}, \ldots, A^{(N)}$

The output of \texttt{TuckerHOSVD} has the following error guarantees [55, 193]. These bounds suggest a less expensive \textit{surrogate loss function} to minimize instead when optimizing the core tensor shape subject to a Tucker decomposition size constraint.

**Theorem 5.3.1.2** ([55, Property 10]; [193, Theorem 10.2]). For any tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ and core shape $r \in [I_1] \times \cdots \times [I_N]$, let the output of $\texttt{TuckerHOSVD}(\mathcal{X}, r)$ be $\mathcal{G} \in \mathbb{R}^{R_1 \times \cdots \times R_N}$ and $A^{(n)} \in \mathbb{R}^{I_n \times R_n}$, for each $n \in [N]$. If we let

$$
\bar{\mathcal{X}}_{\text{HOSVD}}(r) \overset{\text{def}}{=} \mathcal{G} \times_1 A^{(1)} \times_2 \cdots \times_N A^{(N)}
$$

(5.3.1)

denote the reconstructed $r$-truncated tensor, then

$$
\left\| \mathcal{X} - \bar{\mathcal{X}}_{\text{HOSVD}}(r) \right\|_F^2 \leq \sum_{n=1}^N \sum_{i_n = R_n + 1}^{I_n} \left( \sigma^{(n)}_{i_n} \right)^2 \\
\leq N \cdot L(\mathcal{X}, r).
$$

Furthermore, we have $L(\mathcal{X}, r) \leq \left\| \mathcal{X} - \bar{\mathcal{X}}_{\text{HOSVD}}(r) \right\|_F^2$.

Theorem 5.3.1.2 implies that the following function is a meaningful proxy for the reconstruction error of an optimal rank-$r$ Tucker decomposition.

**Definition 5.3.1.3.** Define the surrogate loss of core shape $r$ as

$$
\bar{L}(\mathcal{X}, r) \overset{\text{def}}{=} \sum_{n=1}^N \sum_{i_n = R_n + 1}^{I_n} \left( \sigma^{(n)}_{i_n} \right)^2.
$$

(5.3.2)
To summarize so far, for any core shape $r \in [I_1] \times \cdots \times [I_N]$, we are guaranteed that

$$\frac{1}{N} \cdot \bar{L}(\mathcal{X}, r) \leq L(\mathcal{X}, r) \leq \bar{L}(\mathcal{X}, r).$$

### 5.3.2 Tucker packing problem

Next, observe that the sum of squared singular values across all mode-$n$ unfoldings of $\mathcal{X}$ is

$$\sum_{n=1}^{N} \sum_{i_n=1}^{I_n} \left( \sigma_{(n)}^{i_n} \right)^2 = \sum_{n=1}^{N} \|X_{(n)}\|_F^2 = N \|\mathcal{X}\|_F^2.$$

This means we can solve a singular value packing problem instead by considering the complement of the surrogate loss. The following lemma is a wrapper for the truncated HOSVD error guarantees in Theorem 5.3.1.2.

**Lemma 5.3.2.1.** For any tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ and budget $c \geq 1 + \sum_{n=1}^{N} I_n$ for the size of the Tucker decomposition, let the set of feasible core shapes be

$$F = \{ r \in [I_1] \times \cdots \times [I_N] : \prod_{n=1}^{N} R_n + \sum_{n=1}^{N} I_n R_n \leq c \}.$$

Then, we have

$$\bar{r}^* \overset{\text{def}}{=} \arg \min_{r \in F} \bar{L}(\mathcal{X}, r) = \arg \max_{r \in F} \sum_{n=1}^{N} \sum_{i_n=1}^{I_n} \left( \sigma_{(n)}^{i_n} \right)^2.$$

Further, if $r^* \overset{\text{def}}{=} \arg \min_{r \in F} L(\mathcal{X}, r)$ is an optimal budget-constrained core shape, then

$$L(\mathcal{X}, \bar{r}^*) \leq N \cdot L(\mathcal{X}, r^*).$$

**Proof.** For any $n$, we have

$$\sum_{i_n=1}^{I_n} \left( \sigma_{(n)}^{i_n} \right)^2 = \|X_{(n)}\|_F^2 = \|\mathcal{X}\|_F^2.$$
Therefore, for any choice of \( r = (R_1, R_2, \ldots, R_N) \), we have
\[
\left[ \sum_{n=1}^{N} \sum_{i_{n}=1}^{R_n} \left( \sigma_{i_{n}}^{(n)} \right)^2 \right] + \left[ \sum_{n=1}^{N} \sum_{i_{n}=R_n+1}^{I_n} \left( \sigma_{i_{n}}^{(n)} \right)^2 \right] = N \| \mathcal{F}^\circ \|_F^2.
\]

This is a constant value that only depends on \( \mathcal{F} \), so minimizing \( \overline{L}(\mathcal{F}, r) \) is equivalent to maximizing the packing version since both problems optimize over the same set \( F \).

Lastly, we have
\[
\overline{L} (\mathcal{F}, \overline{r}) \leq \overline{L} (\mathcal{F}, r^*) \leq N \cdot L (\mathcal{F}, r^*),
\]
where the first inequality follows from optimizing the surrogate loss and the second inequality follows from Theorem 5.3.1.2 since that result holds for all core shapes. □

To find a core shape whose optimal Tucker decomposition approximates the optimal loss \( L(\mathcal{F}, r^*) \) subject to a size constraint, we solve the maximization problem described in Lemma 5.3.2.1. Optimizing this proxy objective is substantially less expensive than methods that rely on rank-\( r \) Tucker decomposition solvers as a subroutine. We formalize this idea by introducing the more general problem below.

**Definition 5.3.2.2** (Tucker packing problem). *Given a shape \((I_1, \ldots, I_N) \in \mathbb{Z}_{\geq 1}^N, N\) non-increasing sequences \(a^{(n)} \in \mathbb{R}_{\geq 0}^{I_n}\), and a budget \( c \geq 1\), the Tucker packing problem asks to find a core shape \( r \in [I_1] \times \cdots \times [I_N] \) that solves:

\[
\begin{align*}
\text{maximize} & \quad \sum_{n=1}^{N} \sum_{i_{n}=1}^{R_n} a_{i_{n}}^{(n)} \\
\text{subject to} & \quad \prod_{n=1}^{N} R_n + \sum_{n=1}^{N} I_n R_n \leq c
\end{align*}
\]

We also denote the objective by \( f(r) \) \(\overset{\text{def}}{=} \sum_{n=1}^{N} \sum_{i_{n}=1}^{R_n} a_{i_{n}}^{(n)}\). To prove this problem is NP-hard.
we need the following definition and theorem.

**Definition 5.3.2.3.** Let $N \geq 2$ be an even integer and $w_1, \ldots, w_N \geq 1$ be integers. The *EQUIPARTITION* problem asks to determine whether there exists a subset $S \subseteq [N]$ of size $n/2$ such that

$$\sum_{i \in S} w_i = \sum_{i \in [N] \setminus S} w_i.$$

**Lemma 5.3.2.4 ([203, SP12]).** EQUIPARTITION is NP-complete.

We now give a reduction from the equipartition problem to the Tucker packing problem.

**Theorem 5.3.2.5.** The Tucker packing problem is NP-hard.

**Proof.** Let $T, w_1, \ldots, w_T$ be an instance of EQUIPARTITION where $w_n \geq 2$ for all $n \in [T]$. Notice that the assumption $w_n \geq 2$ is without loss of generality because we can multiply all of the values $w_1, \ldots, w_T$ by two.

Let $M = \sum_{n \in [T]} w_n$ be the sum of all weights, and let $N \geq T$ be the smallest integer such that $2^{N-T/2} > 4(N-T) + 3M/2$. Now we construct an instance of the Tucker packing problem. For each $n \in [T]$, let:

- $I_n = w_n$
- $a_1^{(n)} = 2M$
- $a_2^{(n)} = M + w_n$
- $a_i^{(n)} = 0$, for all $i_n \in [I_n] \setminus \{1, 2\}$

Next, for each $n \in [N] \setminus [T]$, let:

- $I_n = 2$
- $a_1^{(n)} = a_2^{(n)} = 2M$
Finally, set the budget to be \( c = 2^{N-T/2} + 4(N - T) + 3M/2 \).

First, notice that this is a valid instance of the Tucker packing problem since \( a_1^{(n)} \geq a_2^{(n)} \geq \cdots \geq a_{i_n}^{(n)} \) for all \( n \in [N] \). Further, since \( N = O(T \cdot \log_2(3M/2)) \) and \( a_{i_n}^{(n)} = 0 \) for \( i_n \in [I_n] \setminus \{1, 2\} \), the size of the description of this problem is polynomial in the size of the description of the corresponding EQUIPARTITION problem.

Now we consider a decision version of this Tucker packing problem in which we are asked to determine whether there exists a feasible solution \((R_1, \ldots, R_N)\) such that
\[
\sum_{n=1}^{N} \sum_{i_n=1}^{R_n} a_{i_n}^{(n)} \geq M(4N - 3T/2) + M/2. \tag{5.3.5}
\]

We show that a positive answer to the decision version of the Tucker packing problem in (Equation 5.3.5) implies a positive answer to the EQUIPARTITION problem and vice versa.

Suppose the answer to the decision version of the Tucker packing problem is YES, and \( r^* \) is an optimal solution such that
\[
\sum_{n=1}^{N} \sum_{i_n=1}^{R_n^*} a_{i_n}^{(n)} \geq M(4N - 3T/2) + M/2 \quad \text{and} \quad \prod_{n=1}^{N} R_n^* + \sum_{n=1}^{N} I_n R_n^* \leq c.
\]

Since
\[
c = 2^{N-T/2} + 4(N - T) + 3M/2 < 2 \cdot 2^{N-T/2} = 2^{N-T/2+1},
\]
there are at most \( N - T/2 \) values of \( R_n^* \) such that \( R_n^* \geq 2 \). Further, since \( a_{i_n}^{(n)} = 0 \) for all \( i_n \geq 3 \), we never have \( R_n^* > 2 \) in a minimal optimal solution. It follows that \( R_n^* \in \{1, 2\} \) for all \( n \in [N] \), and
\[
\prod_{n=1}^{N} R_n^* \leq 2^{N-T/2}.
\]

Next, we establish the structure of an optimal solution to this Tucker packing instance. Observe that \( \hat{r} = (\hat{R}_1, \ldots, \hat{R}_N) \) with \( \hat{R}_1 = \cdots = \hat{R}_T = 1 \) and \( \hat{R}_{T+1} = \cdots = \hat{R}_N = 2 \) is a feasible solution \( r \) that achieves an objective value of \( M(4N - 2T) \). Now consider any
feasible solution in which there exists \( i \in [T] \) and \( j \in [N] \setminus [T] \) such that \( R_i = 2 \) and \( R_j = 1 \). If we switch the values of \( R_i \) and \( R_j \), then the cost decreases by \( w_i - 2 \geq 0 \) (i.e., the solution is still feasible), and the objective value increases by \( M - w_i > 0 \). Therefore, since \( \mathbf{r} \) is feasible, in an optimal solution we have \( R_n = 2 \) for all \( n \in [N] \setminus [T] \) and at most \( T / 2 \) of the \( R_n \)'s for \( n \in [T] \) are equal to two.

Let \( S = \{ i \in [T] : R_i^* = 2 \} \). Then by construction we have

\[
\sum_{n \in S} a_2^{(n)} = \sum_{n \in S} (M + w_n) = M |S| + \sum_{n \in S} w_n < M(|S| + 1).
\]

Moreover, since the answer to the decision problem is YES and in an optimal solution we have \( R_n = 2 \) for all \( n \in [N] \setminus [T] \), it follows that

\[
\sum_{n \in S} a_2^{(n)} = f(\mathbf{r}^*) - \sum_{n=1}^{N} a_1^{(n)} - \sum_{n=T+1}^{N} a_2^{(n)}
\]

\[
= f(\mathbf{r}^*) - 2NM - 2(N-T)M \geq M(4N-3T/2) + M/2 - 2NM - 2(N-T)M \]

\[
= MT/2 + M/2.
\]

Therefore,

\[
M(|S| + 1) > MT/2 + M/2,
\]

which implies \( |S| > T/2 - 1/2 \), so \( |S| \geq T/2 \) since \( |S| \) and \( T/2 \) are integers. Using the characterization above about an optimal solution together with the fact that the budget is strictly less than \( 2^{N-T/2+1} \) gives us \( |S| \leq T/2 \). Thus, a YES to the decision problem implies that \( |S| = T/2 \), which further implies \( \prod_{n=1}^{N} R_n^* = 2^{N-T/2} \).

It then follows from our choice of budget \( c \) that

\[
\sum_{n=1}^{N} I_n R_n^* \leq c - 2^{N-T/2} = 4(N-T) + 3M/2,
\]

259
which then by the definition of $I_n$ implies that
\[
\left( \sum_{n=1}^{T} w_n R_n^* + \sum_{n=T+1}^{N} 2R_n^* \right) = \left( M + \sum_{n \in S} w_n \right) + 4(N - T) \\
\leq 4(N - T) + 3M/2 \implies \sum_{n \in S} w_n \leq M/2.
\]

Furthermore, using (Equation 5.3.6), the definition of the $a_{i_n}^{(n)}$'s, and the fact that $|S| = T/2$, we have
\[
\sum_{n \in S} a_{i_n}^{(n)} = \sum_{n \in S} (M + w_n) = |S|M + \sum_{n \in S} w_n \geq MT/2 + M/2 \implies \sum_{n \in S} w_n \geq M/2.
\]

Putting everything together, we get $\sum_{n \in S} w_n = M/2$. Therefore, $S$ is a solution for the EQUIPARTITION problem.

Now suppose the answer to the EQUIPARTITION problem is YES. Let $S \subseteq [T]$ such that $|S| = T/2$ and $\sum_{n \in S} w_n = M/2$. Construct $r^*$ as follows: For each $n \in S \cup ([N] \setminus [T])$, set $R_n^* = 2$; for each $n \in [T] \setminus S$, set $R_n^* = 1$.

Then, by the definitions of $I_n$ and $a_{i_n}^{(n)}$ above, we have
\[
\sum_{n=1}^{N} \sum_{i_n=1}^{R_n} a_{i_n}^{(n)} \geq M(4N - 3T/2) + M/2 \quad \text{and} \quad \prod_{n=1}^{N} R_n^* + \sum_{n=1}^{N} I_n R_n^* \leq c,
\]

which completes the proof. \qed

NP-hardness motivates the need for efficient approximation algorithms. In Section 5.4, we develop a \textit{polynomial-time approximation scheme} (PTAS) for the Tucker packing problem. We leave the existence of a fully-polynomial time approximation scheme (FPTAS) as a challenging open question for future works.

To conclude, since Tucker packing is the complement of surrogate loss minimization, we must quantify how a $(1 - \varepsilon)$-approximation for the packing problem can affect the error incurred in the surrogate loss.
Lemma 5.3.2.6. Let $r \in [I_1] \times \cdots \times [I_N]$ be any core shape that achieves a \((1 - \varepsilon/N)\)-approximation to the Tucker packing problem. Then, we have

$$
\text{RRE}(\mathcal{X}, r) \leq N \cdot \text{RRE}(\mathcal{X}, r^*) + \varepsilon,
$$

where $\text{RRE}(\mathcal{X}, r) := L(\mathcal{X}, r) / \|\mathcal{X}\|_F^2$.

Proof. Let $\varepsilon' = \frac{\varepsilon}{N}$ and $\bar{r}^*$ be the optimal shape for the surrogate loss $\bar{L}$. If $r = (R_1, \ldots, R_N)$ is a \((1 - \varepsilon')\)-approximation to the Tucker packing problem, it follows that

$$
\frac{\bar{L}(\mathcal{X}, r)}{\|\mathcal{X}\|_F^2} = \frac{N \|\mathcal{X}\|_F^2 - \Sigma_{n=1}^N \Sigma_{i_n=1}^{R_n} \left(\sigma_{i_n}^{(n)}\right)^2}{\|\mathcal{X}\|_F^2} \leq \frac{N \|\mathcal{X}\|_F^2 - (1 - \varepsilon') \Sigma_{n=1}^N \Sigma_{i_n=1}^{R_n} \left(\sigma_{i_n}^{(n)}\right)^2}{\|\mathcal{X}\|_F^2} = \frac{\bar{L}(\mathcal{X}, \bar{r}^*)}{\|\mathcal{X}\|_F^2} + \varepsilon' \left(N - \frac{\bar{L}(\mathcal{X}, \bar{r}^*)}{\|\mathcal{X}\|_F^2}\right) \leq \frac{\bar{L}(\mathcal{X}, \bar{r}^*)}{\|\mathcal{X}\|_F^2} + \varepsilon.
$$

Theorem 5.3.1.2 gives us $L(\mathcal{X}, r) \leq \bar{L}(\mathcal{X}, r) \leq N \cdot L(\mathcal{X}, r)$. By definition

$$
\bar{L}(\mathcal{X}, \bar{r}^*) \leq \bar{L}(\mathcal{X}, r^*),
$$

so we have

$$
\text{RRE}(\mathcal{X}, r) = \frac{L(\mathcal{X}, r)}{\|\mathcal{X}\|_F^2} \leq \frac{\bar{L}(\mathcal{X}, r)}{\|\mathcal{X}\|_F^2} \leq \frac{\bar{L}(\mathcal{X}, \bar{r}^*)}{\|\mathcal{X}\|_F^2} + \varepsilon
$$

$$
\leq \frac{\bar{L}(\mathcal{X}, r^*)}{\|\mathcal{X}\|_F^2} + \varepsilon \leq N \cdot \frac{L(\mathcal{X}, r^*)}{\|\mathcal{X}\|_F^2} + \varepsilon = N \cdot \text{RRE}(\mathcal{X}, r^*) + \varepsilon,
$$

as desired. \qed

Remark 5.3.2.7. We can obtain global approximation guarantees for Tucker decompo-
sition reconstruction error by (1) finding an approximately optimal core shape, (2) running TuckerHOSVD to initialize the Tucker decomposition, and (3) using alternating least squares (ALS) to improve the tensor decomposition. This is analogous to how k-means++ enhances Lloyd’s algorithm [204].

5.4 Algorithm

5.4.1 Warm-up: Connections to multiple-choice knapsack

To start, consider a simplified version of the Tucker packing problem that only accounts for the size of the core tensor, i.e., the factor matrices do not use any of the budget. We show that after two simple transformations this new problem reduces to the multiple-choice knapsack problem,\(^1\) which is NP-hard [206] but has an FPTAS [207].

Concretely, the optimization problem is

\[
\text{maximize} \quad \sum_{n=1}^{N} \sum_{i_n=1}^{R_n} a_{i_n}^{(n)}
\]

subject to \( \prod_{n=1}^{N} R_n \leq c \)

\[5.4.2\] 

**Prefix sums transformation.** To get closer to a 0-1 knapsack problem, define new coefficients by taking the prefix sums of the \( a_{i_n}^{(n)} \)'s, for each \( n \in [N] \) and \( i_n \in [I_n] \):

\[
p_{i_n}^{(n)} \overset{\text{def}}{=} \sum_{j_n=1}^{i_n} a_{j_n}^{(n)}.
\]

\[1\]The multiple-choice knapsack problem is a 0-1 knapsack problem where the items are partitioned into \( N \) classes and exactly one item must be taken from each class [205].
This “core size-only” Tucker packing problem can be reformulated as the following integer program:

\[
\begin{align*}
\text{maximize} & \quad \sum_{n=1}^{N} \sum_{i_n=1}^{I_n} p_{i_n}^{(n)} x_{i_n}^{(n)} \\
\text{subject to} & \quad \prod_{n=1}^{N} \sum_{i_n=1}^{I_n} i_n x_{i_n}^{(n)} \leq c \\
& \quad \sum_{i_n=1}^{I_n} x_{i_n}^{(n)} = 1 \quad \forall n \in [N] \\
& \quad x_{i_n}^{(n)} \in \{0, 1\} \quad \forall n \in [N], i_n \in [I_n]
\end{align*}
\]

We optimize over \(i_n\) instead of \(R_n\) for notational brevity.

**Log transformation.** Next, replace constraint (Equation 5.4.3) with the linear inequality

\[
\sum_{n=1}^{N} \sum_{i_n=1}^{I_n} \log(i_n) x_{i_n}^{(n)} \leq \log(c).
\]

This substitution is valid because in any feasible solution, for each \(n \in [N]\), exactly one of \(x_1^{(n)}, x_2^{(n)}, \ldots, x_{I_n}^{(n)}\) is equal to one and the rest are zero. Putting everything together, this core size-only Tucker packing problem is the following multiple-choice knapsack problem:

\[
\begin{align*}
\text{maximize} & \quad \sum_{n=1}^{N} \sum_{i_n=1}^{I_n} p_{i_n}^{(n)} x_{i_n}^{(n)} \\
\text{subject to} & \quad \sum_{n=1}^{N} \sum_{i_n=1}^{I_n} \log(i_n) x_{i_n}^{(n)} \leq \log(c) \\
& \quad \sum_{i_n=1}^{I_n} x_{i_n}^{(n)} = 1 \quad \forall n \in [N] \\
& \quad x_{i_n}^{(n)} \in \{0, 1\} \quad \forall n \in [N], i_n \in [I_n]
\end{align*}
\]

**Theorem 5.4.1.1** ([207]). There exists an algorithm that computes a \((1 - \epsilon)\)-approximation
to problem (Equation 5.4.4) in time and space \(O(N^2\varepsilon^{-1}\sum_{n=1}^{N} I_n)\).

The FPTAS in Theorem 5.4.1.1 for multiple-choice knapsack uniformly downscales all coefficients \(p_{i_n}^{(n)}\) in the objective, rounds them, and then uses dynamic programming.

5.4.2 PTAS for the Tucker packing problem

Now we consider the true cost of a Tucker decomposition, i.e., the size of the core tensor and the factor matrices. We first introduce a simple grid-search algorithm that solves approximate Tucker packing for a general class of feasible solutions (i.e., downwards closed sets). This captures the Tucker packing problem and will be useful for extending our results to tree tensor networks in Section 5.5.

**Definition 5.4.2.1.** For any \(N \geq 1\) and \((I_1, \ldots, I_N) \in \mathbb{Z}_{\geq 1}^N\), let \(F \subseteq [I_1] \times \cdots \times [I_N]\). The set \(F\) is downward closed if for any pair \((R_1, \ldots, R_N), (R'_1, \ldots, R'_N) \in [I_1] \times \cdots \times [I_N]\) such that \(R'_n \leq R_n\) for all \(n \in [N]\), \((R_1, \ldots, R_N) \in F\) implies that \((R'_1, \ldots, R'_N) \in F\).

**Lemma 5.4.2.2.** Let \(0 < \varepsilon \leq 1\) and \(F \subseteq [I_1] \times \cdots \times [I_N]\) be downwards closed. For each \(n \in [N]\), define

\[
S_{n}^{(\varepsilon)} = \{\left\lfloor (1 + \varepsilon)^k \right\rfloor : k \in \mathbb{Z}_{\geq 0}, \left\lfloor (1 + \varepsilon)^k \right\rfloor \leq I_n\}.
\]

Let \(r^*\) be an optimal solution to the generalized problem

\[
\text{maximize} \quad \sum_{n=1}^{N} \sum_{i_n=1}^{R_n} a_{i_n}^{(n)} \quad (5.4.5)
\]

subject to \((R_1, \ldots, R_N) \in F\)

and let \(r^{(\varepsilon)} = (R_1^{(\varepsilon)}, \ldots, R_N^{(\varepsilon)})\) be an optimal solution to

\[
\text{maximize} \quad \sum_{n=1}^{N} \sum_{i_n=1}^{R_n} a_{i_n}^{(n)} \quad (5.4.6)
\]

subject to \((R_1, \ldots, R_N) \in \left(S_{1}^{(\varepsilon)} \times \cdots \times S_{N}^{(\varepsilon)}\right) \cap F\)
Then, $f(r^{(\varepsilon)}) \geq (1+\varepsilon)^{-1} f(r^*)$. Further, there is an algorithm that finds an optimal solution of (Equation 5.4.6) with running time $O \left( \sum_{n=1}^{N} I_n + \varepsilon^{-N} \prod_{n=1}^{N} (1 + \log_2(I_n)) \right)$.

**Proof.** Let $k_n \geq 0$ be the largest integer such that $(1 + \varepsilon)^{k_n} \leq R^*_n$ for each $n \in [N]$. Further, let

$$
\widehat{R}_n = \left\lfloor (1 + \varepsilon)^{k_n} \right\rfloor.
$$

Since $R^*_n$ is an integer, we know that $\widehat{R}_n \leq R^*_n$. Therefore, because $F$ is downwards closed, $\widehat{r} = (\widehat{R}_1, \ldots, \widehat{R}_N)$ is a feasible solution to (Equation 5.4.6). It follows that

$$
f(\widehat{r}) \leq f(r^{(\varepsilon)}). \quad (5.4.7)
$$

Now we will show that $f(r^*) \leq (1 + \varepsilon)f(\overline{r})$. Since $a_1^{(n)} \geq \cdots \geq a_I^{(n)} \geq 0$ for all $n \in [N]$, we have

$$
(1 + \varepsilon) \sum_{i_n=1}^{\widehat{R}_n} a_i^{(n)} \geq \sum_{i_n=1}^{\left\lfloor (1+\varepsilon)\widehat{R}_n \right\rfloor} a_i^{(n)}. \quad (5.4.8)
$$

By the definition of $k_n$, it follows that

$$
(1 + \varepsilon)\widehat{R}_n = (1 + \varepsilon) \left\lfloor (1 + \varepsilon)^{k_n} \right\rfloor
\geq (1 + \varepsilon)^{k_n+1}
> R^*_n.
$$

Since $R^*_n$ is an integer, we have $\left\lfloor (1 + \varepsilon)\widehat{R}_n \right\rfloor \geq R^*_n$. Therefore, using (Equation 5.4.8), we have

$$
(1 + \varepsilon) \sum_{i_n=1}^{\widehat{R}_n} a_i^{(n)} \geq \sum_{i_n=1}^{R^*_n} a_i^{(n)}. \quad (5.4.9)
$$

Finally, summing over $n \in [N]$ and using (Equation 5.4.7) gives us

$$
(1 + \varepsilon) f(\overline{r}) \geq f(r^*) \implies f(r^{(\varepsilon)}) \geq (1 + \varepsilon)^{-1} f(r^*).
$$
Algorithm. Now we design and analyze a simple algorithm to solve the grid-search problem in (Equation 5.4.6). First observe that

$$|S_n^{(\varepsilon)}| = 1 + \left\lfloor \log_{1+\varepsilon}(I_n) \right\rfloor = 1 + \left\lfloor \frac{\log_2(I_n)}{\log_2(1+\varepsilon)} \right\rfloor.$$ 

It follows that the number of feasible solutions for Problem (Equation 5.4.6) is

$$O\left(\prod_{n=1}^{N} \left(1 + \frac{\log_2(I_n)}{\log_2(1+\varepsilon)}\right)\right) = O\left(\frac{1}{\log_2(1+\varepsilon)} \prod_{n=1}^{N} (1 + \log_2(I_n))\right),$$

since $\log_2(1+\varepsilon) \leq 1$ for $\varepsilon \leq 1$. Further, observing that $\log_2(1+\varepsilon) \geq \varepsilon$ for $0 < \varepsilon \leq 1$ implies a bound of

$$O\left(\frac{1}{\varepsilon^N} \prod_{n=1}^{n} (1 + \log_2(I_n))\right)$$

on the number of feasible solutions.

After computing the prefix sums for elements of $S_n^{(\varepsilon)}$'s, we can iterate over the elements of $S_1^{(\varepsilon)} \times \cdots \times S_N^{(\varepsilon)}$ in the lexicographical order, check for feasibility, and compute the objective value $f(r)$ for any candidate solution in amortized time $O(1)$. Finally, note that the prefix sums for elements of $S_n^{(\varepsilon)}$ can be computed in $O(I_n)$ time. $\square$

The time complexity in Lemma 5.4.2.2 is exponential in the order of the tensor, so we now focus on designing our main algorithm TuckerPackingSolver, whose running time is $O(\text{poly}(N, \log c, I_n))$ for constant values of $\varepsilon > 0$.

Algorithm description. There are two phases in this algorithm: (1) exhaustively search over all “small” core shapes by trying all budget allocation splits when the factor matrix cost is low; and (2) try coarser splits between the core tensor size and factor matrix costs. In the large phase, we show that it is sufficient to consider $O(\log_{1+\varepsilon}c)$ such splits. Each budget split induces a problem of the form (Equation 5.4.12), which after applying prefix sum and log transformations becomes a familiar 2-dimensional knapsack problem with a
partition matroid constraint:

\[
\text{maximize } \sum_{n=1}^{N} \sum_{i_n=1}^{I_n} p_i^{(n)} x_{i_n}^{(n)} \\
\text{subject to } \sum_{n=1}^{N} \sum_{i_n=1}^{I_n} \log(i_n) x_{i_n}^{(n)} \leq \log(c_{\text{core}}) \\
\sum_{n=1}^{N} \sum_{i_n=1}^{I_n} I_n i_n x_i^{(n)} \leq c - c_{\text{core}} \\
\sum_{i_n=1}^{I_n} x_{i_n}^{(n)} = 1 \quad \forall n \in [N] \\
x_{i_n}^{(n)} \in \{0, 1\} \quad \forall n \in [N], i_n \in [I_n]
\]

More generally, (Equation 5.4.10) is a \(d\)-budgeted matroid independent set problem (see, e.g., [194]), in which a linear objective function is maximized subject to \(d\) knapsack constraints and a matroid constraint. Recall that the multi-dimensional knapsack problem (even without any matroid constraints) does not admit an FPTAS unless \(P = NP\) [208, 209]. It does, however, have a PTAS as shown by the next theorem.

**Theorem 5.4.2.3** ([194, Corollary 4.4]). There is a PTAS (i.e., a \((1 - \varepsilon)\)-approximation algorithm) for the \(d\)-budgeted matroid independent set problem with running time

\[O(m^{O(d^2/\varepsilon)}),\]

where \(m\) is the number of items.

The number of items in (Equation 5.4.10) is \(m = \sum_{n=1}^{N} I_n\), one for each core shape dimension choice. Thus, since \(d = 2\), this gives a running time of \(O\left(\left(\sum_{n=1}^{N} I_n\right)^{O(1/\varepsilon)}\right)\).

This in turn allows us to bound the overall running time of Algorithm 20.

**Remark 5.4.2.4.** We can use integer linear programming solvers for (Equation 5.4.10) instead of [194], but this is possible only because we decouple the core shape cost and the
factor matrix cost, i.e., because of the $c_{\text{factor}}$ and $c_{\text{core}}$ budget splits.

Algorithm 20: TuckerPackingSolver

1. **Input:** shape $(I_1, \ldots, I_N) \in \mathbb{Z}_{\geq 1}^N$, $N$ non-increasing sequences $a^{(n)} \in \mathbb{R}_{\geq 0}$, budget $c \geq 1 + \sum_{n=1}^N I_n$, error $\varepsilon > 0$
2. Initialize $S \leftarrow \emptyset$
3. for $c_{\text{factor}} \in \lceil 1/\varepsilon \rceil \sum_{n=1}^N I_n$ do
   4. Let $r'$ be a $(1 - \varepsilon)$-approximate solution to:
   
   $\begin{align*}
   \text{maximize} & \quad f(r) \\
   \text{subject to} & \quad \prod_{n=1}^N R_n \leq c - c_{\text{factor}} \\
   & \quad \sum_{n=1}^N I_n R_n \leq c_{\text{factor}} \\
   & \quad R_n \in [\min([1/\varepsilon], I_n)] \quad \forall n \in [N]
   \end{align*}$
5. Update $S \leftarrow S \cup \{r'\}$
6. for $k = 0$ to $\lfloor \log_{1+\varepsilon} c \rfloor$ do
7. Set $c_{\text{core}} \leftarrow (1 + \varepsilon)^k$
8. Let $r^{(k)}$ be a $(1 - \varepsilon)$-approximate solution to:
   
   $\begin{align*}
   \text{maximize} & \quad f(r) \\
   \text{subject to} & \quad \prod_{n=1}^N R_n \leq c_{\text{core}} \\
   & \quad \sum_{n=1}^N I_n R_n \leq c - c_{\text{core}} \\
   & \quad R_n \in [I_n] \quad \forall n \in [N]
   \end{align*}$
9. Update $S \leftarrow S \cup \{r^{(k)}\}$
10. return $\arg \max_{r \in S} f(r)$

**Theorem 5.4.2.5.** If $0 < \varepsilon < 1/3$, then Algorithm 20 returns a $(1 - 3 \varepsilon)$-approximate solution to Problem (Equation 5.4.5) in time

$$O\left(\left(\log_{1+\varepsilon}(c) + [1/\varepsilon] \sum_{n=1}^N I_n\right)\left(\sum_{n=1}^N I_n\right)^{O(1/\varepsilon)}\right).$$

**Proof.** TuckerPackingSolver solves for two types of shapes: (1) “small” solutions where each $R_n \leq [1/\varepsilon]$, and (2) “large” solutions where $R_n > [1/\varepsilon]$ for some $n \in [N]$.

In the small phase, observe that since $R_n \leq \min([1/\varepsilon], I_n)$, the factor matrix cost is $\sum_{n=1}^N I_n R_n \leq [1/\varepsilon] \sum_{n=1}^N I_n$. Therefore, we can exhaustively check all small budget splits
of the form $c_{\text{factor}} \in \left[ \left[ \frac{1}{\varepsilon} \right] \sum_{n=1}^{N} I_n \right]$. Each split induces a 2-dimensional knapsack problem with a partition matroid (but for a smaller set of items), so use Theorem 5.4.2.3 to obtain a $(1 - \varepsilon)$-approximation for each subproblem. If an optimal solution $r^*$ to the Tucker packing problem is small, then Algorithm 20 recovers an approximately optimal objective.

For the large phase, assume that the optimal core shape $r^*$ has a dimension $m \in [N]$ such that $R_m^* > \left[ \frac{1}{\varepsilon} \right]$. The algorithm searches over large shapes indirectly by splitting the budget $c$ between the size of the core tensor and the total cost of factor matrices. A crucial observation is that we only need to check $O\left( \log \frac{1}{1 + \varepsilon} c \right)$ different splits because there is a sufficient amount of slack in the large dimension $R_m^*$.

To proceed, let $c_{\text{core}}^* = \prod_{n=1}^{N} R_n^*$, and let $k^*$ be the largest integer such that $(1 + \varepsilon)^k \leq c_{\text{core}}^*$. Define $\tilde{c}_{\text{core}} = (1 + \varepsilon)^{k^*}$, and let $\tilde{r} = (\tilde{R}_1, \ldots, \tilde{R}_N)$ where

$$\tilde{R}_n = \begin{cases} R_n^* & \text{if } n \neq m, \\ \left\lceil R_n^*/(1 + \varepsilon) \right\rceil & \text{if } n = m. \end{cases}$$

Since we assumed $R_m^* \geq \left[ \frac{1}{\varepsilon} \right]$ is a large dimension,

$$\tilde{R}_m = \left\lceil R_m^*/(1 + \varepsilon) \right\rceil \geq \left\lceil \frac{1}{\varepsilon} \right\rceil / (1 + \varepsilon).$$

Next, observe that

$$\prod_{n=1}^{N} \tilde{R}_n \leq \frac{1}{1 + \varepsilon} \prod_{n=1}^{N} R_n^* = \frac{c_{\text{core}}}{1 + \varepsilon} < \frac{(1 + \varepsilon)^{k^*+1}}{1 + \varepsilon} = (1 + \varepsilon)^{k^*} = \tilde{c}_{\text{core}},$$

and $\sum_{n=1}^{N} I_n \tilde{R}_n \leq \sum_{n=1}^{N} I_n R_n^* \leq c - c_{\text{core}}^* \leq c - \tilde{c}_{\text{core}}$. Therefore, $\tilde{r}$ is a feasible solution of (Equation 5.4.12) for $k = k^*$. Furthermore, $f(r^{(k^*)}) \geq (1 - \varepsilon) f(\tilde{r})$.

Next, we show that $f(\tilde{r}) \geq (1 - 2\varepsilon) f(r^*)$. For any $n \neq m$, we have $\tilde{R}_n = R_n^*$, so it
follows that
\[
\sum_{i_n=1}^{R_n} a_{i_n}^{(n)} = \sum_{i_n=1}^{R_n} a_{i_n}^{(n)}.
\]  
(5.4.13)

For the large dimension \(m\), we have
\[
\widehat{R}_m = \left\lfloor \frac{R^*_m}{1 + \varepsilon} \right\rfloor \geq \frac{R^*_m}{1 + \varepsilon} - 1 = \frac{R^*_m - (1 + \varepsilon)}{1 + \varepsilon} \cdot \frac{R^*_m}{R^*_m}.
\]

The assumption \(R^*_m \geq \lceil 1/\varepsilon \rceil\) then gives
\[
\frac{R^*_m - (1 + \varepsilon)}{R^*_m} \geq 1 - \frac{1 + \varepsilon}{\lceil 1/\varepsilon \rceil} \geq 1 - \varepsilon - \varepsilon^2.
\]

It follows for any \(\varepsilon \geq 0\) that
\[
\widehat{R}_m \geq \frac{1 - \varepsilon - \varepsilon^2}{1 + \varepsilon} \cdot \frac{R^*_m}{R^*_m} \geq (1 - 2\varepsilon) \frac{R^*_m}{R^*_m}.
\]

Since \(a_{i_1}^{(m)} \geq \cdots \geq a_{i_m}^{(m)}\) is non-increasing, we have
\[
\sum_{i_m=1}^{\widehat{R}_m} a_{i_m}^{(m)} \geq (1 - 2\varepsilon) \sum_{i_m=1}^{R^*_m} a_{i_m}^{(m)}.
\]  
(5.4.14)

Combining (Equation 5.4.13) and (Equation 5.4.14) gives \(f(\widehat{\mathbf{r}}) \geq (1 - 2\varepsilon) f(\mathbf{r}^*)\), so then
\(f(\mathbf{r}^{(k)}) \geq (1 - \varepsilon)(1 - 2\varepsilon) f(\mathbf{r}^*) \geq (1 - 3\varepsilon) f(\mathbf{r}^*)\), which proves the approximation guarantee.

Finally, the running time follows from our reductions to the 2-dimensional knapsack problem with a partition matroid constraint, and using Theorem 5.4.2.3 for each budget split.  
\(\square\)

To obtain a PTAS, it is enough to consider the budget splits \(c_{\text{factor}} \in [\sum_{n=1}^{N} I_n^2]\), i.e., line 2 of TuckerPackingSolver, but this gives a worse running time as explained in the following.
**Rationale for two different types of budget splits in Algorithm 20.** Instead of considering two different budget splitting methods (i.e., with $c_{\text{factor}}$ and $c_{\text{core}}$), one could just consider the budget splits $c_{\text{factor}} \in [\sum_{n=1}^{N} I_n^2]$ over problems of the form:

$$\text{maximize } f(r)$$

subject to

$$\prod_{n=1}^{N} R_n \leq c - c_{\text{factor}}$$

$$\sum_{n=1}^{N} I_n R_n \leq c_{\text{factor}}$$

$$R_n \in [I_n] \quad \forall n \in [N]$$

This approach also gives a PTAS. The running time, however, is

$$O\left(\left(\sum_{n=1}^{N} I_n^2\right)\left(\sum_{n=1}^{N} I_n\right)^{O(1/\varepsilon)}\right).$$

In contrast, the running time of Algorithm 20 is

$$O\left(\left(\log_{1+\varepsilon}(c) + \lceil 1/\varepsilon \rceil \sum_{n=1}^{N} I_n\right)\left(\sum_{n=1}^{N} I_n\right)^{O(1/\varepsilon)}\right).$$

Note that $c \leq \sum_{n=1}^{N} I_n^2 + \prod_{n=1}^{N} I_n$. Therefore,

$$\log_{1+\varepsilon}(c) = O\left(\frac{1}{\varepsilon} \sum_{n=1}^{N} \log_2(I_n)\right) \ll \lceil 1/\varepsilon \rceil \sum_{n=1}^{N} I_n,$$

where the first equality follows from the concavity of the logarithm function. Thus, our running time is

$$O\left(\left(\lceil 1/\varepsilon \rceil \sum_{n=1}^{N} I_n\right)\left(\sum_{n=1}^{N} I_n\right)^{O(1/\varepsilon)}\right),$$
which is always smaller than
\[
O \left( \sum_{n=1}^{N} I_n^2 \left( \sum_{n=1}^{N} I_n \right)^{O(1/\varepsilon)} \right).
\]

5.5 Tree tensor network decompositions

Here we consider a general decomposition called tree tensor network [210, 191], which includes Tucker decomposition, tensor-train decomposition, and hierarchical Tucker decomposition as special cases.

**Definition 5.5.0.1 (Tree tensor network).** Let \( \mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_N} \) be any tensor. Let \( G = (V, E) \) be a rooted tree with \( N \) leaves where each node \( v \) corresponds to a subset \( S_v \subseteq [N] \). The leaves are the \( N \) singletons of \([N] \), and internal nodes \( v \) are recursively defined by \( S_v = \cup_{u \in C_v} S_u \), where \( C_v \) is the set of children of \( v \).

Each edge \( e \in E \) is endowed an integer \( R_e \geq 1 \). Then, a (truncated) tree tensor network decomposition of \( \mathcal{X} \) for tree \( G \) is the following collection of tensors, each corresponding to a \( v \in V \). For each leaf, the tensor is \( A^{(v)} \in \mathbb{R}^{I_v \times R_e} \), where \( e \) is the edge connecting \( v \) to its parent. For each internal node \( v \), its tensor is \( \mathcal{T}^{(v)} \in \mathbb{R}^{R_{e_1} \times \cdots \times R_{e_k}} \), where \( E_v = \{ e_1, \ldots, e_k \} \) is the set of edges incident to \( v \).

The output tensor \( \mathcal{X} \) is constructed by taking the mode-wise products of all the “node tensors” in \( G \) over their corresponding edges. These products commute and are associative (see, e.g., Proposition 2.17 in [191]).

**Remark 5.5.0.2.** The tree tensor network for Tucker decomposition corresponds to a tree of depth one, and for hierarchical Tucker decomposition it is an (almost) balanced binary tree [210, 211].

**Remark 5.5.0.3.** Although the tree tensor network is considered for general Hilbert spaces and has been defined in full generality using the tensor network notations (see, e.g., [193, 272],
Chapter 11] and [191, Chapter 3], we consider finite-dimensional Euclidean spaces to keep our notation simple.

We now provide a few examples for tree tensor network decompositions.

**Example (Tucker decomposition).** Tucker decomposition corresponds to a tree tensor of depth one. For example, the tree tensor in Fig. 5.2 consists of matrices $A_1 \in \mathbb{R}^{I_1 \times R_{e_1}}, A_2 \in \mathbb{R}^{I_2 \times R_{e_2}}, A_3 \in \mathbb{R}^{I_3 \times R_{e_3}}, A_4 \in \mathbb{R}^{I_4 \times R_{e_4}}$, and tensor $T_v \in \mathbb{R}^{R_{e_1} \times R_{e_2} \times R_{e_3} \times R_{e_4}}$. The corresponding reconstruction is

$$\tau = T_v \times_{e_4} A_4 \times_{e_3} A_3 \times_{e_2} A_2 \times_{e_1} A_1.$$ 

**Example (Hierarchical Tucker decomposition).** we give an example of a tensor of order 4 to better understand Definition 5.5.0.1. Consider the tree illustrated in Fig. 5.3. This tree tensor network corresponds to matrices $A_1 \in \mathbb{R}^{I_1 \times R_{e_1}}, A_2 \in \mathbb{R}^{I_2 \times R_{e_2}}, A_3 \in \mathbb{R}^{I_3 \times R_{e_3}}, A_4 \in \mathbb{R}^{I_4 \times R_{e_4}}$, and tensors $T_u \in \mathbb{R}^{R_{e_1} \times R_{e_2} \times R_{e_5}}, T_v \in \mathbb{R}^{R_{e_3} \times R_{e_4} \times R_{e_6}}, T_w \in \mathbb{R}^{R_{e_5} \times R_{e_6}}$. The corresponding reconstruction is

$$\tau = T_w \times_{e_6} T_v \times_{e_5} T_u \times_{e_4} A_4 \times_{e_3} A_3 \times_{e_2} A_2 \times_{e_1} A_1.$$
Now we generalize the definition of tensor unfolding.

**Definition 5.5.0.4.** For any $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ and $S \subseteq [N]$, the matricization $X_{(S)} \in \mathbb{R}^{P \times Q}$, where $P = \prod_{n \in S} I_n$ and $Q = \prod_{n \in [N] \setminus S} I_n$, is the matrix with the entries of $\mathcal{X}$ arranged lexicographically by their original index tuples.

The next theorem gives a polynomial-time algorithm for finding a tree tensor network decomposition that achieves bounded reconstruction error for specified $R_e$ values.

**Theorem 5.5.0.5 ([211, 191]).** Let $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ and $G = (V, E)$, $R_e$ for $e \in E$ be the tree tensor network parameters as in Definition 5.5.0.1. There exists a polynomial-time algorithm that finds $\mathcal{T}^{(v)}$ for $v \in V$ (for leaves these tensors are the matrices $A^{(v)}$) such that

$$\|\mathcal{X} - \mathcal{T}\|_F^2 \leq \sum_{v \in V \setminus \{r\}} \sum_{i_e = R_v + 1}^{P_v} \left(\sigma_{i_e}^{(v)}\right)^2 \leq (|V| - 1) \cdot \|\mathcal{X} - \mathcal{T}_{\text{best}}\|_F^2,$$

where $\mathcal{T}_{\text{best}}$ is the best tree tensor network decomposition for $G$ and the values $R_e$, $r$ is the root node, $P_v = \prod_{n \in S_v} I_n$, and $\sigma_{i_e}^{(v)}$ is the $i$-th singular value of $X_{(S_v)}$.

Further, the size of the tree tensor network decomposition is $\sum_{v \in L} I_v R_v + \sum_{v \in J} \sum_{e \in E_v} R_e$, where $L$ is the set of leaves, $J$ is the set of internal nodes, and $R_v$ is value on the edge that connects $v$ to its parent.

**Remark 5.5.0.6.** Theorem 5.5.0.8 is implied by Lemma 5.4.2.2.

It follows that we can define the NP-hard tree tensor network packing problem, which generalizes Tucker packing.

**Definition 5.5.0.7 (Tree tensor network packing).** Given a shape $(I_1, \ldots, I_N) \in \mathbb{Z}_{\geq 1}^N$, tree $G = (V, E)$ with leaves $L = [N]$ and internal nodes $J$, $|V| - 1$ non-increasing sequences (corresponding to non-root nodes) $\mathbf{a}^{(v)} \in \mathbb{R}^{P_v \geq 0}$ with $P_v = \prod_{n \in S_v} I_n$, and the budget $c \geq 1$,
Figure 5.4: Comparison of five core shape solvers on four real-world tensors (columns) for increasing values of the Tucker decomposition size budget $c \leq 100,000$. The plots in the top row are the HOSVD Tucker packing objective value $f(r)$ for the core shape solutions $r$, the middle row is the RRE, and the bottom row is the running time of each algorithm in seconds.

The tree tensor network packing problem asks to find $R_v \in \mathbb{Z}_{\geq 1}$ for $v \in V \setminus \{r\}$, where $r$ is the root and $N_v$ is the set of neighbors of node $v$, that solves:

$$\text{maximize} \quad \sum_{v \in V \setminus \{r\}} \sum_{i=1}^{R_v} a_i^{(v)} \quad (5.5.1)$$

$$\text{subject to} \quad \sum_{v \in J} \prod_{u \in N_v} R_u + \sum_{v \in L} I_v R_v \leq c \quad (5.5.2)$$

**Theorem 5.5.0.8.** There is a $(1 - \varepsilon)$-approximation algorithm for the tree tensor network packing problem that runs in time

$$O\left( \sum_{v \in V \setminus \{r\}} P_v + \varepsilon^{-([V]-1)} \prod_{v \in V \setminus \{r\}} (1 + \log_2(P_v)) \right).$$
5.6 Experiments

We compare several algorithms for computing the core shapes of size-constrained Tucker decompositions on four real-world tensors (see Table 5.1). These experiments demonstrate the effectiveness of using the surrogate loss \( \bar{L}(\mathcal{D}, r) \) in place of the true relative reconstruction error (RRE), both in terms of solution quality and running time. All experiments use NumPy [187] with an Intel Xeon W-1235 processor (3.7 GHz, 8.25MB cache) and 128GB of RAM. The source code is available online\(^2\).

Table 5.1: Statistics for tensor datasets used in experiments.

<table>
<thead>
<tr>
<th>TENSOR</th>
<th>SHAPE</th>
<th>SIZE</th>
</tr>
</thead>
<tbody>
<tr>
<td>CARDIAC MRI</td>
<td>256 \times 256 \times 14 \times 20</td>
<td>18,350,080</td>
</tr>
<tr>
<td>HYPERSPECTRAL</td>
<td>1024 \times 1344 \times 33</td>
<td>45,416,488</td>
</tr>
<tr>
<td>VICROADS</td>
<td>1084 \times 2033 \times 96</td>
<td>211,562,112</td>
</tr>
<tr>
<td>COIL-100</td>
<td>7200 \times 128 \times 128 \times 3</td>
<td>353,894,400</td>
</tr>
</tbody>
</table>

5.6.1 Algorithms

The first four algorithms we consider are based on HOSVD Tucker packing: they compute the mode-\( n \) singular values \( \sigma_{i_n}^{(n)} \) and take \( a^{(n)} = \{(\sigma_{i_n}^{(n)})^2\}_{i_n=1}^{I_n} \) as input to their Tucker packing instance. The fifth is a commonly used greedy algorithm that computes true losses \( L(\mathcal{D}, r) \) at each step.

**HOSVD-IP** is the `TuckerPackingSolver` algorithm with \( \epsilon = 0.25 \), but we use the integer programming solver in `scipy.optimize.mlip` to solve each budget split subproblem instead of the PTAS in [194].

**HOSVD-greedy** maximizes the same packing objective by repeating

\[
    r \leftarrow \arg \max_{r' \in N(r)} f(r'),
\]

\(^2\)https://github.com/fahrbach/approximately-optimal-core-shapes
where $N(r)$ is the set of neighboring feasible core shapes $r' = r + e_n$ and $e_n \in \{0, 1\}^N$ is a standard unit vector. This is Algorithm 3.1 in [173] with additional budget constraints.

**HOSVD-bang-for-buck** is analogous to HOSVD-greedy, but it increments the dimension in each step that maximizes $(f(r') - f(r))/(\text{cost}(r') - \text{cost}(r))$, where \text{cost}(r) is the size of the rank-$r$ Tucker decomposition.

**HOSVD-brute-force** exhaustively checks all feasible core shapes and outputs the maximum Tucker packing objective.

**RRE-greedy** constructs the core shape by computing $O(N)$ rank-$r'$ Tucker decompositions in each step and incrementing the dimension that most improves the RRE, Concretely, the update is $r \leftarrow \arg \min_{r' \in N(r)} L(\mathcal{X}, r')$, similar to the Greedy-TL algorithm of [61].

### 5.6.2 Results

We consider the budgets $c \leq 100,000$ for all tensor datasets. For each $c$, we run each algorithm to get core shape $r$. Then in Figure 5.4, we plot the packing objective $f(r)$, the RRE, i.e., $L(\mathcal{X}, r)/\|\mathcal{X}\|_F^2$, and the algorithm running time (including the mode-$n$ singular value computations) as a function of $c$. Each $L(\mathcal{X}, r)$ computation uses 20 iterations of HOOI. We experiment on the following datasets.

- **Cardiac MRI.** $256 \times 256 \times 14 \times 20$ tensor whose elements are MRI measurements indexed by $(x, y, z, t)$, where $(x, y, z)$ is a point in space and $t$ corresponds to time.

- **Hyperspectral.** $1024 \times 1344 \times 33$ tensor of time-lapse hyperspectral radiance images of a nature scene that is undergoing illumination changes [58].

- **VicRoads.** $1084 \times 2033 \times 96$ tensor containing 2033 days of traffic volume data from Melbourne and its surrounding suburbs. This data comes from a network of 1084 road sensors measured in 15 minute intervals [212].

- **COIL-100.** $7200 \times 128 \times 128 \times 3$ tensor containing 7200 colored photos of 100 different objects (72 images per object) taken at 5-degree rotations [140]. This is a
widely-used dataset in the computer vision research community.

**Cardiac MRI** shows that maximizing the HOSVD Tucker packing objective $f(r)$ can give noticeably better RRE than RRE-greedy, while also running 1000x faster. If we take a closer look at the core shapes these algorithms output, HOSVD-\{brute-force, IP\} always return core shapes of the form $(x, y, z, 1)$, whereas the greedy algorithms allocate budget to the time dimension as $c$ increases, e.g., $(x, y, z, 3)$. Increases in the fourth dimension correspond to points of degradation in $f(r)$ and RRE in Figure 5.4.

This tensor is also small enough to see differences in the running times of the HOSVD packing algorithms. In particular, we see (1) that there is a fixed cost for computing the $a_{i_n}^{(n)}$’s, and (2) that HOSVD-\{greedy, bang-for-buck\} are faster than HOSVD-IP, which is faster than HOSVD-brute-force. All algorithms are significantly faster than RRE-greedy.

**Hyperspectral** shows that the surrogate loss $\tilde{L}(D, r)$ and RRE can guide greedy algorithms to the same core shapes, and that HOSVD-greedy can achieve maximum the Tucker packing objective. We see that computing the higher-order singular values becomes the bottleneck for the HOSVD solvers, not solving the packing instances themselves.

**VicRoads** shows a clear gap between RRE and the surrogate loss. While HOSVD-\{greedy, bang-for-buck\} are suboptimal in the packing objective, they achieve the same RRE as the 100x slower RRE-greedy algorithm. This data demonstrates a shortcoming of the surrogate loss, but also shows that higher-order singular values can still be effective.

**COIL-100** is perhaps the most interesting tensor because it shows the *non-monotonic* behavior of greedy HOSVD Tucker packing algorithms. Similar to cardiac MRI, every time a greedy core shape solver increases the dimension of the first index (corresponding to the number of objects), the packing objective $f(r)$. This effect also appears in the RRE plots, but it happens in the opposite direction.
CHAPTER 6

SOCIALLY FAIR $k$-MEANS CLUSTERING

In this chapter, we show that the popular $k$-means clustering algorithm (Lloyd’s heuristic), used for a variety of scientific data, can result in outcomes that are unfavorable to subgroups of data (e.g., demographic groups). Such biased clusterings can have deleterious implications for human-centric applications such as resource allocation. We present a fair $k$-means objective and algorithm to choose cluster centers that provide equitable costs for different groups. The algorithm, Fair-Lloyd, is a modification of Lloyd’s heuristic for $k$-means, inheriting its simplicity, efficiency, and stability. In comparison with standard Lloyd’s, we find that on benchmark datasets, Fair-Lloyd exhibits unbiased performance by ensuring that all groups have equal costs in the output $k$-clustering, while incurring a negligible increase in running time, thus making it a viable fair option wherever $k$-means is currently used.

6.1 Introduction

Clustering, or partitioning data into dissimilar groups of similar items, is a core technique for data analysis. Perhaps the most widely used clustering algorithm is Lloyd’s $k$-means heuristic [213, 13, 214].

Lloyd’s algorithm starts with a random set of $k$ points (“centers”) and repeats the following two-step procedure: (a) assign each data point to its nearest center; this partitions the data into $k$ disjoint groups (“clusters”); (b) for each cluster, set the new center to be the average of all its points. Due to its simplicity and generality, the $k$-means heuristic is widely used across the sciences, with applications spanning genetics [215], image segmentation [216], grouping search results and news aggregation [217], crime-hot-spot detection [218], crime pattern analysis [219], profiling road accident hot spots [220], and market segmentation [221].
Lloyd’s algorithm is a heuristic to minimize the $k$-means objective: choose $k$ centers such that the average squared distance of a point to its closest center is minimized. Note that, these $k$ centers automatically define a clustering of the data simply by assigning each point to its closest center. To better describe the $k$-means objective and Lloyd’s algorithm in the context of human-centric applications, let us consider two applications. In crime mapping and crime pattern analysis, law enforcement would run Lloyd’s algorithm to partition areas of crime. This partitioning is then used as a guideline for allocating patrol services to each area (cluster). Such an assignment reduces the average response time of patrol units to crime incidents. A second application is market segmentation, where a pool of customers is partitioned using Lloyd’s algorithm, and for each cluster, based on the customer profile of the center of that cluster, a certain set of services or advertisements is assigned to the customers in that cluster.

In such human-centric applications, using the $k$-means algorithm in its original form, can result in unfavorable and even harmful outcomes for some demographic groups in the data. To illustrate bias, consider the Adult dataset from the UCI repository [222]. This dataset consists of census information of individuals, including some sensitive attributes such as whether the individuals self-identified as male or female. Lloyd’s algorithm can be executed on this dataset to detect communities and eventually summarize communities with their centers.

Fig. 6.1a shows the average $k$-means clustering cost for the Adult dataset [222] for males vs females. The standard Lloyd’s algorithm results in a clustering that incurs up to 15% higher cost for females compared to males. Fig. 6.1b shows that this bias is even more noticeable among the five different racial groups in this dataset. The average cost for an Asian-Pac-Islander individual is up to 4 times worse than the average cost for a white individual. A similar bias can be observed in the Credit dataset [223] between lower-educated and higher-educated individuals (Fig. 6.1c).

In this chapter, we address the critical goal of fair clustering, i.e., a clustering whose
cost is more equitable for different groups. This is, of course, an important and natural goal, and there has been substantial work on fair clustering, including for the $k$-means objective. Prior work has focused almost exclusively on proportionality, i.e., ensuring that sensitive attributes are distributed proportionally in each cluster [224, 225, 226, 227, 228]. In many application scenarios, including the ones illustrated above, one can view each setting of a sensitive attribute as defining a subgroup (e.g., gender or race), and the critical objective is the cost of the clustering for each subgroup: are one or more groups incurring a significantly higher average cost?

In light of this consideration, we consider a different objective. Rather than minimizing the average clustering cost over the entire dataset, the objective of socially fair $k$-means is to find a $k$-clustering that minimizes the maximum of the average clustering cost across different (demographic) groups, i.e., minimizes the maximum of the average $k$-means objective applied to each group.

*Can social fairness be achieved efficiently, while preserving the simplicity and generality of standard $k$-means algorithm?*

Applying existing algorithms for fair clustering with proportionality constraints leads to poor solutions for social fairness (see Fig. 6.11 for comparison on standard datasets), so we need a different solution. Our objective is similar to the recent line of work on minmax fairness through multi-criteria optimization [66, 71, 229].
6.1.1 Our results

We answer the above question affirmatively, with an algorithm we call Fair-Lloyd. Similar to Lloyd’s algorithm, it is a two-step iteration with the only difference being how the centers are updated: (a) assign each data point to its nearest center to form clusters (b) choose $k$ new fair centers such that the maximum average clustering cost across different demographic groups is minimized. This step is particularly easy for $k$-means — average the points in each cluster. We prove that, the fair centers can also be computed efficiently: using a simple one-dimensional line search when the data consists of two (demographic) groups, and using standard convex optimization algorithms when the data consists of more than two groups. Furthermore, when the data consists of two groups, the convergence of our algorithm is independent of the original dimension of the data and the number of clusters.

We prove convergence, stability and approximability guarantees and apply our method to multiple real-world clustering tasks. The results show clearly that Fair-Lloyd generates a clustering of the data with equal average clustering cost for individuals in different demographic groups. Moreover, its computational cost remains comparable to Lloyd’s method. Each iteration, to find the next set of centers, is a convex optimization problem and can be implemented efficiently using Gradient Descent. For two groups, we give a line-search method which is significantly faster. This extends to a fast heuristic for $m > 2$ groups whose distance to optimality can be tracked. This approach might be of independent interest as a very efficient heuristic for similar optimization problems.

Due to the simplicity and efficiency of the Fair-Lloyd algorithm, we suggest it as an alternative to the standard Lloyd’s algorithm in human-centric and other subgroup-sensitive applications where social fairness is a priority.

6.1.2 Fair k-means: Objective and Algorithm

To introduce the fair $k$-means objective, we define a more general notion: the $k$-means cost of a set of points $U$ with respect to a set of centers $C = \{c_1, \ldots, c_k\}$ and a partition
\( \mathcal{U} = \{U_1, \ldots, U_k\} \) of \( U \) is

\[
\Delta(C, \mathcal{U}) := \sum_{i=1}^{k} \sum_{p \in U_i} ||p - c_i||^2.
\]

For a set of centers \( C \), let \( \mathcal{U}_C \) be a partition of \( U \) such that if \( p \in U_i \) then \( ||p - c_i|| = \min_{1 \leq j \leq k} ||p - c_j|| \). Then the standard \( k \)-means objective is

\[
\min_{C = \{c_1, \ldots, c_k\}} \Delta(C, \mathcal{U}_C),
\]

i.e., to find a set of \( k \) centers \( C = \{c_1, \ldots, c_k\} \) that minimizes \( \Delta(C, \mathcal{U}_C) \).

For an illustrative example of the potential bias for different subgroups of data, see Fig. 6.2 left. The two centers selected by minimizing the \( k \)-means objective are both close to one subgroup, and therefore the other subgroup has higher average cost. Note that the notion of fairness based on proportionality also prefers this clustering which imposes a higher average cost on the purple subgroup. To introduce our fair \( k \)-means objective and algorithm, in this section we focus on the case of two (demographic) groups. In Section 6.3, we discuss how to generalize our framework to more than two groups.

The fair \( k \)-means objective for two groups \( A, B \) such that \( U = A \cup B \) is the larger average cost:

\[
\Phi(C, \mathcal{U}) := \max \left\{ \frac{\Delta(C, \mathcal{U} \cap A)}{|A|}, \frac{\Delta(C, \mathcal{U} \cap B)}{|B|} \right\},
\]

where \( \mathcal{U} \cap A = \{U_1 \cap A, \ldots, U_k \cap A\} \). The goal of fair \( k \)-means is to minimize \( \Phi(C, \mathcal{U}_C) \), so as to minimize the higher average cost. As illustrated in Fig. 6.2 right, minimizing this objective results in a set of centers with equal average cost to individuals of different groups. In fact, as we will soon see, the solution to this problem equalizes the average cost of both groups in most cases. Next we present the fair \( k \)-means algorithm (or Fair-Lloyd) in Algorithm 21.

The second step of each iteration uses a minimization procedure to assign centers fairly.
Figure 6.2: Two demographic groups are shown with blue and purple. The 2-means objective minimizing the average clustering cost prefers the clustering (and centers) shown in the left figure. This clustering incurs a much higher average clustering cost for purple than for blue. The clustering in the right figure has more equitable clustering cost for the two groups.

**Algorithm 21: Fair-Lloyd**

1. **Input:** A set of points $U = A \cup B$, and $k \in \mathbb{N}$
2. Initialize the set of centers $C = \{c_1, \ldots, c_k\}$.
3. **repeat**
   4. 1. Assign each point to its nearest center in $C$ to form a partition $U = \{U_1, \ldots, U_k\}$ of $U$.
   5. 2. Pick a set of centers $C$ that minimizes $\Phi(C, U)$.
   
   \[ C \leftarrow \text{Line Search}(U, U) \]

6. **until** convergence;
7. **return** $C = \{c_1, \ldots, c_k\}$

...to a given partition of the data. While this can be done via a gradient descent algorithm, we show in Section 6.2 that it can be solved very efficiently using a simple line search procedure (see Algorithm 22) due to the structure of fair centers (Section 6.2.1). In Section 6.2.3, we discuss some other properties of the fair $k$-means and Fair-Lloyd (Algorithm 21). More specifically, we discuss the stability of the solution found by Fair-Lloyd, the convergence of Fair-Lloyd, and approximation algorithms that can be used for fair $k$-means (e.g., to initialize the centers). In summary, our fair version of the $k$-means inherits its attractive properties while making the objective and outcome more equitable to subgroups of data.
6.1.3 Related Work

\textbf{k-means objective and Lloyd’s algorithm.} The \textit{k}-means objective is NP-hard to optimize [230] and even NP-hard to approximate within a factor of \((1 + \epsilon)\) [231]. The best known approximation algorithm for the \textit{k}-means problem finds a solution within a factor \(\rho + \epsilon\) of optimal, where \(\rho \approx 6.357\) [232]. The running time of Lloyd’s algorithm can be exponential even on the plane [233].

As for the quality of the solution found, Lloyd’s heuristic converges to a local optimum [234], with no worst-case guarantees possible [235]. It has been shown that under certain assumptions on the existence of a sufficiently good clustering, this heuristic recovers a ground truth clustering and achieves a near-optimal solution to the \textit{k}-means objective function [236, 237, 238]. For all the difficulties with the analysis, and although many other techniques has been proposed over the years, Lloyd’s algorithm is still the most widely used clustering algorithm in practice [239].

\textbf{Fairness.} During the past years, machine learning has seen a huge body of work on fairness. Many formulations of fairness have been proposed for supervised learning and specifically for classification tasks [12, 240, 241, 242]. The study of the implications of bias in unsupervised learning started more recently [224, 243, 244, 66, 245, 246]. We refer the reader to [247] for a summary of proposed definitions and algorithmic advances.

Majority of the literature on fair clustering have focused on the proportionality/balance of the demographical representation inside the clusters [224] — a notion much in the nature of the widely known \textit{disparate impact} doctrine. Proportionality of demographical representation has initially been studied for the \textit{k}-center and \textit{k}-median problems when the data comprises of two demographic groups [224], and later on for the \textit{k}-means problem and for multiple demographic groups [225, 248, 228, 227]. Among other notions of fairness in clustering, one could mention proportionality of demographical representation in the set of cluster centers [246] or in large subsets of a cluster [249].

Our proposed notion of a fair clustering is different and comes from a broader viewpoint
on fairness, aiming to enforce any objective-based optimization task to output a solution with *equitable objective value* for different demographic groups. Such an objective-based fairness notion across subgroups could be defined subjectively e.g., by equalizing misclassification rate in classification tasks [250] or by minimizing the maximum error in dimensionality reduction or classification [66, 71, 229]. We define a *socially fair clustering* as the one that minimizes the maximum average clustering cost over different demographic groups. To the best of our knowledge, our work is the first to study fairness in clustering from this viewpoint.

### 6.2 An Efficient Implementation of Fair $k$-Means

The Fair-Lloyd algorithm (Algorithm 21) is a two-step iteration, where the second step is to find a fair set of centers with respect to a partition. A set of centers $C^*$ is fair with respect to a partition $U$ if $C^* = \arg\min_C \Phi(C, U)$. In this section, we show that a simple line search algorithm can be used to find $C^*$ efficiently.

#### 6.2.1 Structure of Fair Centers

We start by illustrating some properties of fair centers. A partition of the data induces a partition of each of the two groups, and hence a set of means for each group. Formally, for a set of points $U = A \cup B$ and a partition $U = \{U_1, \ldots, U_k\}$ of $U$, let $\mu_i^A$ and $\mu_i^B$ be the mean of $A \cap U_i$ and $B \cap U_i$ respectively for $i \in [k]$. Our first observation is that the fair center of each cluster must be on the line segment between the means of the groups induced in the cluster.

**Lemma 6.2.1.1.** Let $U = A \cup B$ and $U = \{U_1, \ldots, U_k\}$ be a partition of $U$. Let $C = (c_1, \ldots, c_k)$ be a fair set of centers with respect to $U$. Then $c_i$ is on the line segment connecting $\mu_i^A$ and $\mu_i^B$.

**Proof.** For the sake of contradiction, assume that there exists an $i \in [k]$ such that $c_i$ is not on the line segment connecting $\mu_i^A$ and $\mu_i^B$. Note that (see [235])
∑_{p \in A \cap U_i} ||p - c_i||^2 = \sum_{p \in A \cap U_i} ||p - \mu_i^A||^2 + |A \cap U_i||\mu_i^A - c_i||^2 (6.2.1)

\sum_{p \in B \cap U_i} ||p - c_i||^2 = \sum_{p \in B \cap U_i} ||p - \mu_i^B||^2 + |B \cap U_i||\mu_i^B - c_i||^2

Let \( c_i' \) be the projection of \( c_i \) to the line segment connecting \( \mu_{A_i} \) and \( \mu_{B_i} \). Then by Pythagorean theorem for convex sets, we have

\[ ||\mu_i^A - c_i||^2 \geq ||\mu_i^A - c_i'||^2 + ||c_i' - c_i||^2 \]
\[ ||\mu_i^B - c_i||^2 \geq ||\mu_i^B - c_i'||^2 + ||c_i' - c_i||^2 \]

Therefore since \( ||c_i' - c_i||^2 > 0 \), we have \( ||\mu_i^A - c_i'|| < ||\mu_i^A - c_i|| \) and \( ||\mu_i^B - c_i'|| < ||\mu_i^B - c_i|| \). Thus, replacing \( c_i \) with \( c_i' \) decreases the fair k-means objective. □

The above lemma implies that, in order to find a fair set of centers, we only need to search the intervals \([\mu_{A_i}, \mu_{B_i}]\). Therefore we can find a fair set of centers by solving a convex program. The following definition will be convenient.

**Definition 6.2.1.2.** Given \( U = A \cup B \) and a partition \( \mathcal{U} = \{U_1, \ldots, U_k\} \) of \( U \), for \( i = 1, \ldots, k \), let

\[ \alpha_i = \frac{|A \cap U_i|}{|A|}, \quad \beta_i = \frac{|B \cap U_i|}{|B|} \text{ and } l_i = ||\mu_i^A - \mu_i^B||. \]

Also let \( M^A = \{\mu_1^A, \ldots, \mu_k^A\} \) and \( M^B = \{\mu_1^B, \ldots, \mu_k^B\} \).

Since \([\mu_i^A, \mu_i^B]\) is a line segment, for the \( i \)'th cluster, we only need to find the distance \( x_i \) of its center \( c_i \) from \( \mu_i^A \). Then the distance of the center from \( \mu_i^B \) is \( l_i - x_i \) where \( l_i \) is the length of the line segment \([\mu_i^A, \mu_i^B]\). By (Equation 6.2.1), the average cost of group \( A \) with respect to a partition \( \mathcal{U} \) in terms of \( x_i \)'s is

\[
\frac{\sum_{i=1}^{k} \sum_{p \in A \cap U_i} ||p - \mu_i^A||^2}{|A|} + \frac{\sum_{i=1}^{k} |A \cap U_i||\mu_i^A - c_i||^2}{|A|}.
\]
\[
\frac{\Delta(M^A, U \cap A)}{|A|} + \sum_{i=1}^{k} \alpha_i x_i^2
\]

Similarly, the average cost of group \( B \) is

\[
\frac{\Delta(M^B, U \cap B)}{|B|} + \sum_{i=1}^{k} \beta_i (l_i - x_i)^2
\]

Hence, our goal is to solve the following optimization problem.

\[
\max \left\{ \frac{\Delta(M^A, U \cap A)}{|A|} + \sum_{i=1}^{k} \alpha_i x_i^2, \frac{\Delta(M^B, U \cap B)}{|B|} + \sum_{i=1}^{k} \beta_i (l_i - x_i)^2 \right\}
\]

subject to \( 0 \leq x_i \leq l_i, \forall i \in [k] \). Note that this is a convex program because the maximum of two convex functions is a convex function. One can see that the point on the line segment \([\mu^A_i, \mu^B_i]\) that has distance \( x_i \) from \( \mu^A_i \) is \( \frac{(l_i-x_i)\mu^A_i + x_i \mu^B_i}{l_i} \), where \( l_i \) is the length of \([\mu^A_i, \mu^B_i]\).

Using a standard trick to write the objective function as a linear function, we can state the problem as the following.

**Corollary 6.2.1.3.** Let \( U = \{U_1, \ldots, U_k\} \) be a partition of \( U = A \cup B \). Then \( C = \{c_1, \ldots, c_k\} \) is a fair set of centers with respect to \( U \) if

\[
c_i = \frac{(l_i-x_i^*)\mu^A_i + x_i^* \mu^B_i}{l_i},
\]

where \((x_1^*, \ldots, x_k^*, \theta^*)\) is an optimal solution to the following convex program.

\[
\min \theta \quad \text{subject to} \quad \frac{\Delta(M^A, U \cap A)}{|A|} + \sum_{i \in [k]} \alpha_i x_i^2 \leq \theta \\
\frac{\Delta(M^B, U \cap B)}{|B|} + \sum_{i \in [k]} \beta_i (l_i - x_i)^2 \leq \theta \\
0 \leq x_i \leq l_i, \forall i \in [k]
\]
We can solve this convex program with standard convex optimization methods such as gradient descent. However, as we show in the next section, we can solve it with a much faster algorithm.

6.2.2 Computing Fair Centers via Line Search

We first need to review a couple of facts about subgradients. For a convex continuous function \( f \), we say that a vector \( u \) is a subgradient of \( f \) at point \( x \) if \( f(y) \geq f(x) + u^T(y-x) \) for any \( y \). We denote the set of subgradients of \( f \) at \( x \) by \( \partial f(x) \).

**Fact 6.2.2.1.** Let \( f \) be a convex function. Then point \( x^* \) is a minimum for \( f \) if and only if \( \bar{0} \in \partial f(x^*) \).

**Fact 6.2.2.2.** Let \( f_1, \ldots, f_m \) be smooth functions and

\[
F(x) = \max_{j \in [m]} f_j(x).
\]

Let \( S_x = \{ j \in [m] : f_j(x) = F(x) \} \). Then the set of subgradients of \( F \) at \( x \) is the convex hull of union of the subgradients of \( f_j \)'s at \( x \) for \( j \in S_x \).

Let

\[
f_A(x) := \frac{\Delta(M^A, \mathcal{U} \cap A)}{|A|} + \sum_{i \in [k]} \alpha_i x_i^2,
\]

\[
f_B(x) := \frac{\Delta(M^B, \mathcal{U} \cap B)}{|B|} + \sum_{i \in [k]} \beta_i (l_i - x_i)^2.
\]

Then we can view the convex program (Equation 6.2.2) as minimizing

\[
f(x) := \max \{f_A(x), f_B(x)\} \text{ s.t. } 0 \leq x_i \leq l_i, \forall i \in [k]. \tag{6.2.3}
\]

Note that \( f(x) \) is convex since the maximum of two convex functions is convex. Therefore by Fact 6.2.2.1, our goal is to find a point \( x^* \) such that \( \bar{0} \in \partial f(x^*) \). Note that \( f_A \) and \( f_B \)
are differentiable. Hence by Fact 6.2.2.2, we only need to look at points \( x \) for which there exists a convex combination of \( \nabla f_A(x) \) and \( \nabla f_B(x) \) that is equal to \( \vec{0} \). As we will see, this set of points is only a one-dimensional curve in \( [0, l_1] \times \cdots \times [0, l_k] \). When \( f_A(x) > f_B(x) \), \( f(x) \) has a unique gradient and it is equal to \( \nabla f_A(x) \). Similarly, when \( f_A(x) < f_B(x) \), we have \( \nabla f(x) = \nabla f_B(x) \). By Fact 6.2.2.2, in the case that \( f_A(x) = f_B(x) \), for any \( \gamma \in [0, 1] \),

\[
u(\gamma, x) := \gamma \nabla f_A(x) + (1 - \gamma) \nabla f_B(x)\]

is a subgradient of \( f(x) \) — and these are the only subgradients of \( f \) at \( x \). Now consider the set

\[Z := \{ x \in [0, l_1] \times \cdots \times [0, l_k] : \exists \gamma \in [0, 1], u(\gamma, x) = \vec{0} \} .\]

In words, set \( Z \) is the set of all points for which there exist a convex combination of gradients of \( f_A \) and \( f_B \) that is equal to \( \vec{0} \). If we find \( x^* \in Z \) such that \( f_A(x^*) = f_B(x^*) \) then any convex combination of \( \nabla f_A(x^*) \) and \( \nabla f_B(x^*) \) is a subgradient of \( f \) at \( x^* \) and therefore \( \vec{0} \in \partial f(x^*) \). Hence \( x^* \) is an optimal solution. We first describe \( Z \) and show that there exists an optimal solution in \( Z \).

**Lemma 6.2.2.3.** Let \( \gamma \in [0, 1] \) and \( u(\gamma, x) = \vec{0} \). Then \( x_i = \frac{(1 - \gamma)\beta_i l_i}{\gamma \alpha_i + (1 - \gamma) \beta_i} \).

**Proof.** We have \( \frac{\partial}{\partial x_i} f_A(x) = 2\alpha_i x_i \) and \( \frac{\partial}{\partial x_i} f_B(x) = 2\beta_i (x_i - l_i) \). Using the fact that \( u(\gamma, x) = \vec{0} \), we have

\[
\gamma (2\alpha_i x_i) + (1 - \gamma) (2\beta_i (x_i - l_i)) = 0.
\]

Hence,

\[
x_i = \frac{(1 - \gamma)\beta_i l_i}{\gamma \alpha_i + (1 - \gamma) \beta_i}.
\]

\( \Box \)

The previous lemma gives a complete description of set \( Z \). One example of set \( Z \) is shown in Fig. 6.3 left for the case of \( k = 2 \). The following is an immediate result of Lemma 6.2.2.3.
Lemma 6.2.2.4. \( Z = \{ x : x_i = \frac{(1-\gamma)\beta_i l_i}{\gamma \alpha_i + (1-\gamma)\beta_i}, \gamma \in [0,1] \} \).

Figure 6.3: Left: an example of the one-dimensional curve for \( k = 2 \). Right: the functions \( f_A \) and \( f_B \) with respect to \( \gamma \), and two steps of the line search algorithm. We can use a line search to find the optimal value of \( \gamma \) and an optimal solution to (Equation 6.2.3).

Note that when \( \gamma = 1 \), this recovers the all-zero vector and when \( \gamma = 0, x = (l_1, \ldots, l_k) \). Therefore these extreme points are also in \( Z \). As we mentioned before if there exists \( x^* \in Z \) such that \( f_A(x^*) = f_B(x^*) \), then \( x^* \) is an optimal solution. Therefore suppose such an \( x^* \) does not exist. One can see that

\[
\frac{d}{d\gamma} \left( \frac{(1-\gamma)\beta_i l_i}{\gamma \alpha_i + (1-\gamma)\beta_i} \right) = \frac{-\alpha_i \beta_i l_i}{(\gamma \alpha_i + (1-\gamma)\beta_i)^2}.
\]

Therefore, for \( 1 \leq i \leq k, x_i \) is decreasing in \( \gamma \). Also one can see that \( f_A \) is increasing in \( x_i \) and \( f_B \) is decreasing in \( x_i \). Therefore \( f_A \) is decreasing in \( \gamma \) and \( f_B \) is increasing in \( \gamma \). Fig. 6.3 right shows an example that illustrates the change of \( f_A \) and \( f_B \) with respect to \( \gamma \). This implies that if there does not exist any \( x^* \in Z \) such that \( f_A(x^*) = f_B(x^*) \), then either \( f_A(\bar{0}) > f_B(\bar{0}) \) or \( f_B(\ell) > f_A(\ell) \), where \( \ell = (l_1, \ldots, l_k) \). In the former case, the optimal solution to (Equation 6.2.2) is \( \bar{0} \) which means that the fair centers are located on the means of points for group \( A \). In the latter case the optimal solution is \( \ell \) which means that the fair centers are located on the means of points for group \( B \). In these extreme cases, there does not exist a set of centers with equal average cost with respect to the particular chosen partition of points.
The above argument asserts that we only need to search the set \( Z \) to find an optimal solution. Each element of \( Z \) is uniquely determined by the corresponding \( \gamma \in [0, 1] \). Our goal is to find an element \( x^* \in Z \) such that \( f_A(x^*) = f_B(x^*) \). Since \( f_A \) is decreasing in \( \gamma \) and \( f_B \) is increasing in \( \gamma \), we can use line search to find such a point in \( Z \). If such a point does not exist in \( Z \), then the line search converges to \( \gamma = 0 \) or \( \gamma = 1 \). Two steps of such a line search are shown in Fig. 6.3. See Algorithm 22 for a precise description. Using this line search algorithm, we can solve the convex program described in (Equation 6.2.3) to \( \epsilon \) error in \( O(\sum_{i=1}^{k} \log \frac{L}{\epsilon}) \) time.

**Algorithm 22: Line Search\((U, U)\)**

1. **Input:** A set of points \( U = A \cup B \) and a partition \( \mathcal{U} = \{U_1, \ldots, U_k\} \) of \( U \).
2. Compute \( \alpha_i, \beta_i, \mu_i^A, \mu_i^B, l_i, M_A, M_B \) // See Definition 6.2.1.2.
3. \( \gamma \leftarrow 0.5 \)
4. for \( i = 1, \ldots, T \) do
5. \( x_i \leftarrow \frac{(1-\gamma)\beta_i l_i}{\gamma\alpha_i + (1-\gamma)\beta_i \gamma}, \) for \( i = 1, \ldots, k \)
6. Compute \( f_A(x) \) and \( f_B(x) \)
7. if \( f_A(x) > f_B(x) \) then
8. \( \gamma \leftarrow \gamma + (1/2)^{-(i+1)} \)
9. else if \( f_A(x) < f_B(x) \) then
10. \( \gamma \leftarrow \gamma - (1/2)^{-(i+1)} \)
11. else
12. break
13. \( c_i \leftarrow \frac{(l_i-x_i)\mu_i^A + x_i\mu_i^B}{l_i}, \) for all \( i = 1, \ldots, k \)
14. return \( C = (c_1, \ldots, c_k) \)

### 6.2.3 Fair \( k \)-means is well-behaved

In this section, we discuss the stability, convergence, and approximability of Fair-Lloyd for 2 groups. As we will show in Section 6.3, these results can be extended to \( m > 2 \) groups.

**Stability.** The line search algorithm finds the optimal solution to (Equation 6.2.2). This means that for a fixed partition of the points (e.g., the last clustering that the algorithm outputs), the returned centers are optimal in terms of the maximum average cost of the groups. However, one important question is whether we can improve the cost for the group with the smaller average cost. The following proposition shows that this is not possible;
assuring that the solution is pareto optimal.

**Proposition 6.2.3.1.** Let $x^*$ be the optimal solution for a fixed partition $\mathcal{U} = \{U_1, \ldots, U_k\}$ of $U$. Then there does not exist any other optimal solution with an average cost better than $f_A(x^*)$ or $f_B(x^*)$ for groups $A$ and $B$, respectively.

**Proof.** Let $y$ be another optimal solution. Without loss of generality, suppose $f_A(x^*) \geq f_B(x^*)$. If $f_A(x^*) > f_B(x^*)$, then by our discussion on the line search algorithm, $x^* = \bar{0}$ and it is the only optimal solution. Therefore $y = x^*$. Now suppose $f_A(x^*) = f_B(x^*)$.

For the sake of contradiction and without loss of generality, assume $f_A(x^*) = f_A(y)$, but $f_B(x^*) > f_B(y)$. Therefore $f_A(y) > f_B(y)$. First note that $y \neq \bar{0}$ because if $f_A(\bar{0}) > f_B(\bar{0})$, then for any other $x$ in the feasible region, $f_A(x) > f_B(x)$ which is a contradiction because $f_A(x^*) = f_B(x^*)$. Hence we can decrease one of the coordinates of $y$ by a small amount to get a point $y'$ in the feasible region. If the change is small enough, we have $f_A(y') > f_A(y') > f_B(y') > f_B(y)$ but this is a contradiction because it implies $f(x^*) = f(y) > f(y')$ which means $x^*$ was not an optimal solution. \[\square\]

**Convergence.** Lloyd’s algorithm for the standard $k$-means problem converges to a solution in finite time, essentially because the number of possible partitions is finite [13]. This also holds for the Fair-Lloyd algorithm for the fair $k$-means problem. Note that for any fixed partition of the points, our algorithm finds the optimal fair centers. Also, note that there are only a finite number of partitions of the points. Therefore, if our algorithm continues until a step where the clustering does not change afterward, then we say that the algorithm has converged and indeed the solution is a local optimum. However, note that, in the case where there is more than one way to assign points to the centers (i.e., there exists a point that have more than one closest center), then we should exhaust all the cases, otherwise the output is not necessarily a local optimal. This is not a surprise because the same condition also holds for the Lloyd’s algorithm for the $k$-means problem. For example, see Fig. 6.4. Adjacent points have unit distance from each other. The centers are optimum
for the illustrated clustering. However, they do not form a local optimum because moving \( c_2 \) and \( c_3 \) to the left by a small amount \( \epsilon \) decreases the \( k \)-means objective from 2 to
\[
2(1 - \epsilon)^2 + \epsilon^2 = 2 - 4\epsilon + 3\epsilon^2 < 2.
\]

![Figure 6.4: An example of \( k \)-means problem where the current clustering is not a local optimal and we need to check all the possible partitions with the current centers. \( c_1, c_2, c_3 \) are the centers and the points are marked with the letter A on top of them.](image)

**Initialization.** An important consideration is how to initialize the centers. While a random choice is often used in practice for the \( k \)-means algorithm, another choice that has better provable guarantees [237] is to use a set of centers with objective value that is within a constant factor of the minimum. We will show that a \( c \)-approximation for the \( k \)-means problem implies a \( 2c \)-approximation for the fair \( k \)-means problem, and so this method could be used to initialize centers for Fair-Lloyd as well. The best known approximation algorithm for the \( k \)-means problem finds a solution within a factor \( \rho + \epsilon \) of optimal, where \( \rho \approx 6.357 \) [232].

**Theorem 6.2.3.2.** If the \( k \)-means problem admits a \( c \)-approximation in polynomial time then the fair \( k \)-means problem admits a \( 2c \)-approximation in polynomial time.

**Proof.** Let
\[
g(C) = \frac{\Delta(C, \mathcal{U}_C \cap A)}{|A|} + \frac{\Delta(C, \mathcal{U}_C \cap B)}{|B|}.
\]
This is basically the \( k \)-means objective when we consider a weight of \( \frac{1}{|A|} \) for the points in \( A \) and a weight of \( \frac{1}{|B|} \) for the points in \( B \).

Let \( O \) be an optimal solution to \( g \) and \( S \) be a \( c \)-approximation solution to \( g \) (i.e., \( g(S) \leq \)).
Moreover

$$
\Phi(S, U_S) = \max \left\{ \frac{\Delta(S, U_S \cap A)}{|A|}, \frac{\Delta(S, U_S \cap B)}{|B|} \right\}
$$

$$
\leq \frac{\Delta(S, U_S \cap A)}{|A|} + \frac{\Delta(S, U_S \cap B)}{|B|}
$$

$$
= g(S).
$$

Hence \( \Phi(S, U_S) \leq cg(O) \). Now let \( O' \) be an optimal solution for \( \Phi \). Then

$$
g(O') = \frac{\Delta(O', U_{O'} \cap A)}{|A|} + \frac{\Delta(O', U_{O'} \cap B)}{|B|}
$$

$$
\leq 2 \max \left\{ \frac{\Delta(O', U_{O'} \cap A)}{|A|}, \frac{\Delta(O', U_{O'} \cap B)}{|B|} \right\}
$$

$$
= 2 \Phi(O', U_{O'}) \leq 2 \Phi(S, U_S).
$$

Also by optimality of \( O \) for \( g \), we have \( g(O) \leq g(O') \). Therefore

$$
g(O) \leq 2 \Phi(O', U_{O'}) \leq 2 \Phi(S, U_S) \leq 2 cg(O).
$$

This implies that \( \frac{\Phi(S, U_S)}{\Phi(O', U_{O'})} \leq 2c \). ☐

### 6.3 Generalization to \( m > 2 \) groups

Let \( U = A_1 \cup \cdots \cup A_m \). Then the objective of fair \( k \)-means for \( m \) demographic groups is to find a set of centers \( C \) that minimizes the following

$$
\Phi(C, U_C) := \max \left\{ \frac{\Delta(C, U_C \cap A_1)}{|A_1|}, \ldots, \frac{\Delta(C, U_C \cap A_m)}{|A_m|} \right\},
$$

Let \( U = \{ U_1, \ldots, U_k \} \) be a partition of \( U \), and \( \mu_i^j \) be the mean of \( U_i \cap A_j \) (i.e., the mean of members of subgroup \( j \) in cluster \( i \)). Then by a similar argument to Lemma 6.2.1.1, one can conclude that for a fair set of centers \( C = \{ c_1, \ldots, c_k \} \) with respect to \( U \), \( c_i \) is in the convex
hull of \{\mu^1_i, \ldots, \mu^m_i\}. Then we can generalize the convex program in (Equation 6.2.2) to \(m\) demographic groups as the following:

\[
\begin{align*}
\min_{\theta} & \quad \theta \\
\text{s.t.} & \quad \frac{\Delta(M^j, U \cap A_j)}{|A_j|} + \sum_{i \in [k]} \alpha_i^j ||c_i - \mu^j_i||^2 \leq \theta, \forall j \in [m] \\
& \quad c_i \in \text{Conv}(\mu^1_i, \ldots, \mu^m_i), \forall i \in [k]
\end{align*}
\]

where \(\alpha_i^j = \frac{|U_i \cap A_j|}{|A_j|}\) and \(M^j = \{\mu^j_1, \ldots, \mu^j_k\}\). In (Equation 6.3.1), the same standard trick (as for the case of two groups) is used to make the objective function linear and \(\frac{\Delta(M^j, U \cap A_j)}{|A_j|} + \sum_{i \in [k]} \alpha_i^j ||c_i - \mu^j_i||^2\) is the average cost of group \(j\). The set of \(c_i\)'s found by solving the above convex program will be a fair set of centers with respect to \(U\). We can solve this using standard convex optimization algorithms including gradient descent. However, similar to the case of two groups, we can find a fair set of centers by searching a standard \((m-1)\)-simplex. Namely, we only need to search the following set to find a fair set of centers.

\[
Z = \{C = (c_1, \ldots, c_k) : c_i = \frac{\gamma_1 \alpha_i^1 + \cdots + \gamma_m \alpha_i^m \mu^j_i}{\sum_{j=1}^{m} \gamma_j}, \sum_{j=1}^{m} \gamma_j = 1\}
\]

The following notations will be convenient. For \(C = (c_1, \ldots, c_k)\), and \(j \in [m]\), let

\[
f_j(C) = \frac{\Delta(M^j, U \cap A_j)}{|A_j|} + \sum_{i \in [k]} \alpha_i^j ||c_i - \mu^j_i||^2,
\]

and \(F(C) = \max_{j \in [m]} f_j(C)\). Then the convex program represented in (Equation 6.3.1) is equivalent to \(\min F(C) : c_i \in \text{Conv}(\mu^1_i, \ldots, \mu^m_i), \forall i \in [k]\). Similar to the case of two groups, set \(Z\) is the set of points for which there exist a convex combination of the gradients of \(f_j\)'s that is equal to \(\tilde{0}\).
Let \( u^i_j = c_i - \mu^i_j \). For a vector \( v \), let \( v(s) \) denote its \( s \)’th component. Then we have

\[
\|c_i - \mu^i_i\|^2 = \sum_{s=1}^{d} u^i_i(s)^2
\]

**Theorem 6.3.0.1.** Any optimum solution of (Equation 6.3.1) is in \( Z \).

**Proof.** We can view a set of centers as a point in a \( k \times d \) dimensional space. Let \( \{e_{i,s} : i \in [k], s \in [d]\} \) be the set of standard basis of this space. Then we have

\[
\frac{d}{de_{i,s}} f_j(C) = 2\alpha^j_i u^i_i(s).
\]

By Fact 6.2.2.1 and Fact 6.2.2.2, we only need to show that set \( Z \) is the set of all points for which there exists a convex combinations of \( \nabla f_1(C), \ldots, \nabla f_m(C) \) that is equal to \( \vec{0} \). Let \( 0 \leq \gamma_1, \ldots, \gamma_m \leq 1 \) such that \( \sum_{i=1}^{m} \gamma_i = 1 \). We want to find a \( C \) such that \( \sum_{j=1}^{m} \gamma_j \nabla f_j(C) = \vec{0} \). Therefore for each \( i, s \), we have

\[
0 = \sum_{j=1}^{m} \gamma_j \frac{d}{de_{i,s}} f_j(C) = \sum_{j=1}^{m} \gamma_j (2\alpha^j_i u^i_i(s)) = \sum_{j=1}^{m} 2\gamma_j \alpha^j_i (c_i(s) - \mu^i_i(s)).
\]

Thus,

\[
c_i(s) = \sum_{j=1}^{m} \frac{\gamma_j \alpha^j_i}{\gamma_1 \alpha^j_1 + \cdots + \gamma_m \alpha^j_m} \mu^j_i(s),
\]

and

\[
c_i = \sum_{j=1}^{m} \frac{\gamma_j \alpha^j_i}{\gamma_1 \alpha^j_1 + \cdots + \gamma_m \alpha^j_m} \mu^j_i.
\]

This shows that the set of centers that satisfy \( \sum_{j=1}^{m} \gamma_j \nabla f_j(C) = 0 \), for some \( \gamma_1, \ldots, \gamma_m \), are exactly the members of \( Z \). \( \square \)

Note that any element in \( Z \) is identified by a point in the standard \((m-1)\)-simplex, i.e., \((\gamma_1, \ldots, \gamma_m)\) such that \( \sum_{j=1}^{m} \gamma_j = 1 \). However, the function defined on the \((m-1)\)-simplex is not necessarily convex. Indeed, as shown in Fig. 6.5, it is not even quasiconvex. Thus, one can either use standard convex optimization algorithms to solve the original
convex program in (Equation 6.3.1), or other heuristics to only search the set $Z$. For our experiments, we use a variant of the multiplicative weights update algorithm on set $Z$ — see Algorithm 23.

To certify the optimality of the solution, one can use Fact 6.2.2.1 and show that $\bar{0}$ is a subgradient. However, the iterative algorithms usually do not find the exact optimum, but rather converge to the optimum solution. To evaluate the distance of a solution from the optimum, we propose a min/max theorem for set $Z$ in Section 6.3.2. This theorem allows us to certify that the solutions found by our heuristic in the experiments are within a distance of 0.01 from the optimal.

6.3.1 Multiplicative Weights Update Heuristic

Note that the original optimization problem given in (Equation 6.3.1) is convex. However, we can use a heuristic to solve the problem in the $\gamma$ space. One such heuristic is the multiplicative weights update algorithm [251], precisely defined as Algorithm 23.
Algorithm 23: Multiplicative Weights Update

1. **Input:** Integers \( m \) and \( k \), numbers \( \alpha_i^j \) for \( i \in [k] \) and \( j \in [m] \), and \( \frac{\Delta(M_j, \mathcal{A}_j)}{|A_j|} \) for \( j \in [m] \).
2. \( \gamma_j \leftarrow \frac{1}{m} \) for \( j \in [m] \).
3. for \( t = 1, \ldots, T \) do
4. \( c_i \leftarrow \sum_{j=1}^{m} \gamma_j \alpha_i^j \) for \( i \in [k] \)
5. \( C \leftarrow (c_1, \ldots, c_k) \)
6. Compute \( f_j(C) \) for all \( j \in [m] \)
7. \( F(C) \leftarrow \max_{j \in [m]} f_j(C) \)
8. \( d_j \leftarrow F(C) - f_j(C) \), for \( j \in [m] \)
9. \( \gamma_j \leftarrow \gamma_j \left(1 - \frac{d_j}{\sqrt{t \max_{j \in [m]} d_j}}\right) \)
10. Normalize \( \gamma_j \)’s such that \( \sum_{j=1}^{m} \gamma_j = 1 \)
11. return \( C \)

6.3.2 Certificate of Optimality

Next, we give a min/max theorem that can be used to find a lower bound for the optimum value. Using this theorem, we can certify that, in practice, the multiplicative weights update algorithm finds a solution very close to the optimum.

**Theorem 6.3.2.1.** Let \( S \subseteq [m] \) and

\[
Z_S = \{ C = (c_1, \ldots, c_k) : c_i = \sum_{j=1}^{m} \frac{\gamma_j \alpha_i^j}{\gamma_1 \alpha_1^j + \cdots + \gamma_m \alpha_m^j} \mu_i^j, \sum_{j=1}^{m} \gamma_j = 1, \text{ and } \gamma_j = 0, \forall j \notin S \}.
\]

Then, \( \min_{C \in Z_S} \max_{j \in S} f_j(C) \leq \min_{C \in Z_S} \max_{j \in S} f_j(C) \).

Moreover, \( \min_{C \in Z_S} \max_{j \in S} f_j(C) \leq \min_{C \in [m]} \max_{j \in [m]} f_j(C) \).

**Proof.** Let \( C, C' \in Z_S \) and let \( \gamma, \gamma' \) be the corresponding parameters for \( C, C' \), respectively.

Note that \( Z_S \subseteq Z \). Therefore \( \sum_{j \in [m]} \gamma_j \nabla f_j(C) = \vec{0} \). Hence because \( \gamma_j = 0 \) for any \( j \notin S \), we have \( \sum_{j \in S} \gamma_j \nabla f_j(C) = \vec{0} \). Hence \( (\sum_{j \in S} \gamma_j \nabla f_j(C)) \cdot (C' - C) = 0 \). Therefore there
exists a $j^* \in S$ such that $\nabla f_{j^*}(C) \cdot (C' - C) \geq 0$. Thus because $f_{j^*}$ is convex, we have

$$f_{j^*}(C') \geq f_{j^*}(C) + \nabla f_{j^*}(C) \cdot (C' - C) \geq f_{j^*}(C).$$

Therefore $\max_{j \in S} f_j(C') \geq \min_{j \in S} f_j(C)$. Note that this holds for any $C, C' \in Z$ and this implies the first part of the theorem.

Let $C'$ be the optimum solution to $\min_{C \in Z} \max_{j \in S} f_j(C)$. By Theorem 6.3.0.1, this is an optimum solution to the problem of finding a fair set of centers for the groups in $S$. Moreover, for any set of centers $C''$ outside the convex hull of the centers of groups in $S$, we have $\max_{j \in S} f_j(C') \leq \max_{j \in S} f_j(C'')$ — proof of this is similar to Lemma 6.2.1.1. Hence $\max_{j \in S} f_j(C') \leq \min_{C \in Z_{[m]}} \max_{j \in S} f_j(C)$. Thus we have

$$\max \min_{C \in Z} f_j(C) \leq \min_{C \in Z} \max_{j \in S} f_j(C) = \max_{j \in S} f_j(C') \leq \min_{C \in Z_{[m]}} \max_{j \in S} f_j(C).$$

\qed

Note that we can use Theorem 6.3.2.1, to get a lower bound on the optimum solution of the convex program in (Equation 6.3.1). For example, suppose $C'$ is a solution returned by a heuristic. Then

$$\min_{j \in [m]} f_j(C') \leq \max_{C \in Z_{[m]}} \min_{j \in [m]} f_j(C) \leq \min_{C \in Z_{[m]}} \max_{j \in [m]} f_j(C).$$

Therefore $\min_{j \in [m]} f_j(C')$ is a lower bound for the optimum solution. Hence the difference of the solution returned by the heuristic with the optimum solution is at most

$$(\max_{j \in [m]} f_j(C')) - (\min_{j \in [m]} f_j(C')).$$

This will be very useful for the case where $f_j(C^*) = F(C^*)$ for all $j \in [m]$, where
$C^*$ is the optimum solution. The reason is that in this case, Theorem 6.3.2.1 implies $\max_{C \in Z_{[m]}} \min_{j \in [m]} f_j(C) = \min_{C \in Z_{[m]}} \max_{j \in [m]} f_j(C)$. However this might not be the case and we might have $f_j(C^*) < F(C^*)$ for some $j$. In this case we can use $S \subset [m]$. For example an $S$ that gives a larger lower bound and for which $\max_{C \in Z_{S}} \min_{j \in S} f_j(C) = \min_{C \in Z_{S}} \max_{j \in S} f_j(C)$.

6.3.3 Stability and Approximability

We conclude this section by a discussion on the stability and the approximability of fair $k$-means for $m$ groups.

Our stability results generalizes to $m$ demographic groups. Let $C^* = \{c^*_1, \ldots, c^*_k\}$ be an optimal solution, and $S \subseteq [m]$. Also let $f_j(C^*) = \max_{i \in [m]} f_i(C^*)$ for $j \in S$, and $f_j(C^*) < \max_{i \in [m]} f_i(C^*)$ for $j \not\in S$. Then one can see that, for all $i \in [k]$, $c^*_i \in \text{Conv} \{\mu^j_i : j \in S\}$). This uniquely determines the location of the optimal solution, and thus we cannot improve the value of functions $f_j$ where $j \not\in S$. Moreover, with an argument similar to Proposition 6.2.3.1, one can deduce that we cannot improve the value of functions $f_j$ where $j \in S$.

Moreover, the Fair-Lloyd algorithm for $m$ demographic groups converges to a solution in finite time, essentially because the number of possible partitions of points is finite. Finally, if the $k$-means problem admits a $c$-approximation then the fair $k$-means problem for $m$ demographic groups admits an $mc$-approximation — the proof is similar to the proof of Theorem 6.2.3.2.

6.4 Experimental Evaluation

We consider a clustering to be fair if it has equal clustering costs across different groups. We compare the average clustering cost for different demographic groups on multiple benchmark datasets, using Lloyd’s algorithm and Fair-Lloyd algorithm (Code is accessible on-
Figure 6.6: Average clustering cost of different groups when using Fair-Lloyd algorithm versus the standard Lloyd’s. Rows correspond to different pre-processing methods and columns to the datasets. Note that the fair clustering costs for the two groups are identical or nearly identical in all datasets.

We used three datasets: 1) Adult dataset [222], consists of records of 48842 individuals collected from census data, with 103 features. The demographic groups considered are female/male for the 2-group setting and five racial groups of “Amer-Indian-Eskim”, “Asian-Pac-Islander”, “Black”, “White”, and “Other” for the multiple-groups setting; 2) Labeled faces in the wild (LFW) dataset [226], consists of 13232 images of celebrities. The size of each image is $49 \times 36$ or a vector of dimension 1764. The demographic groups are female/male; and 3) Credit dataset [223], consists of records of 30000 individuals with

\footnote{https://github.com/samirasamadi/SociallyFairKMeans}
21 features. We divided the multi-categorical education attribute to “higher educated” and “lower educated”, and used these as the demographic groups.

As different features in any dataset have different units of measurements (e.g., age versus income), it is standard practice to normalize each attribute to have mean 0 and variance 1. We also converted any categorical attribute to numerical ones. For both Lloyd’s and Fair-Lloyd we tried 200 different center initialization, each with 200 iterations. We used random initial centers (starting both algorithms with the same centers in each run).

For clustering high-dimensional datasets with $k$-means, Principal Component Analysis (PCA) is often used as a pre-processing step [252, 237], reducing the dimension to $k$. We evaluate Fair-Lloyd both with and without PCA. Since PCA itself could induce representational bias towards one of the (demographic) groups, Fair-PCA [66] has been shown to be an unbiased alternative, and we use it as a third pre-processing option. We refer to these three pre-processing choices as w/o PCA, w/ PCA, and w/ Fair-PCA respectively.

**Results.** Fig. 6.6 shows the average clustering cost for different demographic groups. In the first row, all datasets are evaluated in their original dimension with no pre-processing applied (w/o PCA). In the second and third rows (w/ PCA and w/ Fair-PCA), the PCA/Fair-PCA dimension is equal to the target number of clusters $k$.

Our first observation is that the standard Lloyd’s algorithm results in a significant gap between the clustering cost of individuals in different groups, with higher clustering cost for females in the Adult and LFW datasets, and for lower-educated individuals in the Credit dataset. The average clustering cost of a female is up to 15% (11%) higher than a male in the Adult (LFW) dataset when using standard Lloyd’s. A similar bias is observed in the Credit dataset, where Lloyd’s leads up to 12% higher average cost for a lower-educated individual compared to a higher-educated individual.

Our second observation is that the Fair-Lloyd algorithm effectively eliminates this bias by outputting a clustering with equal clustering costs for individuals in different demo-
graphic groups. More precisely, for the Credit and Adult datasets the average costs of two demographic groups are identical, represented by the yellow line in Fig. 6.6. For the LFW dataset, we observe a very small difference in the average clustering cost over the two groups in the fair clustering (0.4%, 1% and 0.6% difference for without PCA, with PCA, and with Fair-PCA respectively). Notably, Fair-Lloyd mitigates the bias of the output clustering independent of whether it is applied on the original data space, on the PCA space, or on the Fair-PCA space. In Fig. 6.8, we show a snapshot of the performance of Fair-Lloyd versus Lloyd’s on the Adult dataset for all three different pre-processing choices.

Fig. 6.7 shows the maximum ratio of average cost between any two racial groups in the Adult dataset, which comprised of five racial groups “Amer-Indian-Eskim”, “Asian-Pac-Islander”, “Black”, “White”, and “Other”. Note that, the max cost ratio of one indicates that all groups have the same average cost in the output clustering. As we observe, the standard Lloyd algorithm results in a significant gap between the cost of different groups resulting in a high max cost ratio overall. As for the Fair-Lloyd algorithm, as the number of clusters increases, it outputs a clustering of the data with same average cost for all the demographic groups.

The price of fairness. Does requiring fairness come at a price, in terms of either running time or overall $k$-means cost? Fig. 6.9 shows the running time of Lloyd’s versus Fair-Lloyd for 200 iterations. Running time for all three datasets is measured in the $k$-dimensional PCA space, where $k$ is the number of clusters. As we observe, Fair-Lloyd incurs a very small overhead in the running time, with only 4%, 4%, and 8% increase (on average over $k$) for the Adult, Credit, and LFW dataset respectively. Moreover, as illustrated in Fig. 6.10, the convergence rate of Lloyd and Fair-lloyd are essentially the same in practice. Finally, the increase in the standard $k$-means cost of Fair-Lloyd solutions (averaged over the entire population ) was at most 4.1%, 2.2% and 0.3% for the LFW, Adult, and Credit datasets, respectively. Arguably, this is outweighed by the benefit of equal cost to the two groups.
Socially fair versus proportionally fair. The first introduced notion of fairness for k-means clustering considered the proportionality of the sensitive attributes in each cluster [224]. We emphasize that improving the proportionality is at odds with improving the maximum average cost of the groups. This can be seen in Fig. 6.2. To illustrate this more, we compared our method to one of the proposed methods that guarantees the proportionality of the clusters on the credit and adult datasets. We used the code provided in [228]. As illustrated in Fig. 6.11, the proportionally fair method fails to achieve an equal average cost for different populations and our methods do not achieve proportionally fair clusters.

6.5 Discussion

Fairness is an increasingly important consideration for Machine Learning, including classification and clustering. Our work shows that the most popular clustering method, Lloyd’s algorithm, can be made fair, in terms of average cost to each subgroup, with minimal increase in the running time or the overall average k-means cost, while maintaining its simplicity, generality and stability. Previous work on fair clustering focused on proportional representation of sensitive attributes within clusters, while we optimize the maximum cost to subgroups. As Fig. 6.2 suggests, and Fig. 6.11 shows on benchmark data sets, these criteria lead to different solutions. We believe that both perspectives are important, and the choice of which clustering to use will depend on the context and application, e.g., proportional representation might be paramount for partitioning electoral precincts, while minimizing cost for every subgroup is crucial for resource allocation.
Figure 6.7: Adult dataset: The maximum ratio of average clustering cost between any two racial groups: “Amer-Indian-Eskim”, “Asian-Pac-Islander”, “Black”, “White”, and “Other”.

Figure 6.8: Adult dataset: comparison of the standard Lloyd’s and Fair-Lloyd algorithm for the three different pre-processing choices of w/o PCA, w/ PCA, and w/ Fair-PCA.

Figure 6.9: Running time (seconds) of Fair-Lloyd algorithm versus the standard Lloyd’s algorithm on the $k$-dimensional PCA space for 200 iterations.
Figure 6.10: Convergence rate of Fair-Lloyd algorithm versus the standard Lloyd’s algorithm for $k = 10$. The plotted objective value for the standard Lloyd is the average cost of clustering over the whole population, and the objective value for Fair-Lloyd is the maximum average cost of the demographic groups. The reported objective values are averaged over 20 runs and the shaded areas are the standard deviations.

Figure 6.11: Comparison of socially fair $k$-means (Fair-Lloyd) to proportionally fair $k$-means (Fairlet) on the Credit and Adult dataset in terms of proportionality and clustering cost.
CHAPTER 7
SCALABLE CONSTANT-FACTOR APPROXIMATION ALGORITHMS FOR
SOCIALLY FAIR $k$-CLUSTERING

In this chapter, we study approximation algorithms for the socially fair $(\ell_p, k)$-clustering problem with $m$ groups which include the socially fair $k$-median ($p = 1$) and $k$-means ($p = 2$). We present (1) a polynomial-time $(5 + 2\sqrt{6})^p$-approximation with at most $k + m$ centers (2) a $(5 + 2\sqrt{6} + \epsilon)^p$-approximation with $k$ centers in time $(nk)^{2O(p)m^2/\epsilon}$, and (3) a $(15 + 6\sqrt{6})^p$ approximation with $k$ centers in time $k^m \cdot \text{poly}(n)$. The former is obtained by a refinement of the iterative rounding method via a sequence of linear programs. The latter two are obtained by converting a solution with up to $k + m$ centers to one with $k$ centers by sparsification methods for (2) and via an exhaustive search for (3). In addition, we discuss how to speed up our algorithm using a combination of coresets and $k$-means++. We compare the performance of our algorithms with existing approximation algorithms on benchmark datasets, and find that our algorithms outperform existing methods.

7.1 Introduction

Automated decision-making using machine learning algorithms is widely adopted in modern society. Examples of real-world decisions made by ML algorithms are numerous and encompass applications with significant societal effects, such as automated content moderation [253] and recidivism prediction [254]. Consequently, there is a need to design new machine learning algorithms that incorporate societal considerations. Fairness is a crucial aspect of these considerations, as discrimination based on seven protected attributes, including race, is prohibited under the Civil Rights Act of 1968. Due to these concerns and the increasing use of algorithms and machine learning in areas such as policing, criminal sentencing, and lending, subfields such as algorithmic fairness and fair optimization have
emerged as important components of modern machine learning [12, 3, 255].

The facility location problem is a well-studied problem in combinatorial optimization. Well-known instances of this problem include the \( k \)-means, \( k \)-median, and \( k \)-center problems. In these cases, the input consists of a finite metric, and the goal is to find \( k \) points (centers or facilities) such that a function based on the distances of each given point to its nearest center is minimized. For \( k \)-means, the objective is to minimize the average squared distance to the nearest center; for \( k \)-median, it is the average distance; and for \( k \)-center, it is the maximum distance. All of these instances can be encompassed by the \((\ell_p, k)\)-clustering problem, which is defined as follows: given a set of clients \( A \) of size \( n \), a set of candidate facility locations \( F \), and a metric \( d \), the task is to find a subset \( F \subseteq F \) of size \( k \) that minimizes \( \sum_{i \in A} d(i, F)^p \), where \( d(i, F) = \min_{j \in F} d(i, j) \). This problem is NP-hard for all values of \( p \) and also hard to approximate [256, 257]. An algorithm providing a \( 2^{O(p)} \)-approximation was given by [258]. Currently, the best approximation factors achieved for the \( k \)-median and \( k \)-means problems on general metrics are a \((2.675 + \epsilon)\)-approximation [259] and a \((9 + \epsilon)\)-approximation [260, 261], respectively.

Here we consider the socially fair extensions of the \((\ell_p, k)\)-clustering problem, where \( m \) different (not necessarily disjoint) subgroups, \( A = A_1 \cup \cdots \cup A_m \), are provided among the data. The objective is to minimize the maximum cost over the groups, ensuring that a common solution does not impose excessive expense on any one of them. Each group can be a subset of the data or any nonnegative weighting.

The goal is to minimize the maximum weighted cost among the groups, expressed as:

\[
\min_{F \subseteq F : |F| = k} \max_{s \in [m]} \sum_{i \in A_s} w_s(i) d(i, F)^p. \tag{7.1.1}
\]

A weighting of \( w_s(i) = 1/|A_s| \) for \( i \in A_s \) corresponds to averaging the groups. These groups often arise from sensitive attributes, such as race and gender, which are protected

---

\( ^{1} \)In some works, the \( p \)'th root of the objective is considered, resulting in different approximation factors in such cases.
against discrimination under the Civil Rights Act of 1968 [63, 64].

The socially fair $k$-median and $k$-means cases correspond to $p = 1$ and $p = 2$ respectively, as introduced by [72, 74]. As discussed and experimentally verified in [72], the objective of the socially fair $k$-means promotes a more equitable average clustering cost among different groups.

The objective function of socially fair $k$-median was initially studied by [262], who provided an $O(\log m + \log n)$-approximation algorithm. Furthermore, the existing approximation algorithms for vanilla $k$-means and $k$-median can be utilized to find $O(m)$-approximate solutions for the socially fair versions [72, 74]. The proof technique directly yields a $m \cdot 2^{O(p)}$-approximation for socially fair $(\ell_p, k)$-clustering. The natural linear programming (LP) relaxation of the socially fair $k$-median problem has an integrality gap of $\Omega(m)$ [74].

More recently, [75] strengthened the LP relaxation of the socially fair $(\ell_p, k)$-clustering by a sparsification technique. The stronger LP has an integrality gap of $\Omega(\log m / \log \log m)$ and their rounding algorithm (similar to [258]) finds a $(2^{O(p)} \log m / \log \log m)$-approximation algorithm for the socially fair $(\ell_p, k)$-clustering. For the socially fair $k$-median, this is asymptotically the best possible in polynomial time under the assumption that $\text{NP} \not\subseteq \cap_{\delta > 0} \text{DTIME}(2^{n^{\delta}})$ [263]. Due to this hardness result, it is natural to consider a bicriteria approximation, which allows for more centers whose total cost is close to the optimal cost for $k$ centers. For the socially fair $k$-median and $0 < \epsilon < 1$, [74] presents an algorithm that gives at most $k/(1 - \epsilon)$ centers with objective value at most $2^{O(p)/\epsilon}$ times the optimum for $k$ centers. Our first result is an improved bicriteria approximation algorithm for the socially fair $\ell_p$ clustering problem with only $m$ additional centers ($m$ is usually a small constant).

**Theorem 7.1.0.1.** There is a polynomial-time bicriteria approximation algorithm for the socially fair $(\ell_p, k)$-clustering problem with $m$ groups that finds a solution with at most $k + m$ centers of cost at most $(5 + 2\sqrt{6})^p \approx 9.9^p$ times the optimal cost for a solution with
Goyal and Jaiswal [264] show that a solution to the socially fair \((\ell_p, k)\)-clustering problem with \(k' > k\) centers and cost \(C\) can be converted to a solution with \(k\) centers and cost at most \(3^{p-1}(C + \text{opt})\) by simply taking the \(k\)-subset of the \(k'\) centers of lowest cost. A proof is included in Section 7.2 for completeness — see Lemma 7.2.0.5. We improve this factor using a sparsification technique.

**Theorem 7.1.0.2.** For any \(\epsilon > 0\), there is a \((5 + 2\sqrt{6} + \epsilon)^p\)-approximation algorithm for the socially fair \((\ell_p, k)\)-clustering problem that runs in time \((nk)^{2^{O(m^2)} + \epsilon}\); there is a \((15 + 6\sqrt{6})^p\)-approximation algorithm that runs in time \(k^m \cdot \text{poly}(n)\).

This raises the question of whether a faster-constant-factor approximation is possible. [264] show under the Gap-Exponential Time Hypothesis\(^2\), it is hard to approximate socially fair \(k\)-median and \(k\)-means within factors of \(1 + 2/e - \epsilon\) and \(1 + 8/e - \epsilon\), respectively, in time \(g(k) \cdot n^{f(m) - o(k)}\), for \(f, g : \mathbb{R}^+ \rightarrow \mathbb{R}^+\); socially fair \((\ell_p, k)\)-clustering is hard to approximate within a factor of \(3^p - \epsilon\) in time \(g(k) \cdot n^{o(k)}\). They also give a \((3 + \epsilon)^p\)-approximation in time \((k/e)^{O(k)} \cdot \text{poly}(n/e)\). This leaves open the possibility of a constant-factor approximation in time \(f(m)\text{poly}(n,k)\).

For the case of \(p \rightarrow \infty\), the problem reduces to fair \(k\)-center problem if we take \(p^{th}\) root of the objective. The problem is much better understood and widely studied along with many generalization [265, 266, 75]. [75] result implies an \(O(1)\)-approximation in this case.

We compare the performance of our bicriteria algorithm against [74] and our algorithm with exactly \(k\) centers against [75] on three different benchmark datasets. Our experiments show that our algorithms consistently outperform these in practice (Section 7.6) and often select fewer centers than the algorithm of [74] (Section 7.6.5).

\(^2\)Informally Gap-ETH states that there is no \(2^{o(n)}\)-time algorithm to distinguish between a satisfiable formula and a formula that is not even \((1 - \epsilon)\) satisfiable.
In terms of running time in practice, our experiments indicate that the bulk of the time is spent on solving the LP relaxation of the problem. Therefore our algorithm is almost as fast as LP-based approximation algorithms for the vanilla $k$-means problem while it gives a more equitable outcome for the subgroups of the population. To speed up our algorithm, in Section 7.5, we show how coresets [267] and the $k$-means++ algorithm [204] can be combined with a result regarding the best $k$-clustering of a bicriteria approximation, to reduce the size of the problem to $O(k \cdot m \cdot \log n)$ datapoints while preserving the optimal solution up to constant factors. This then allows us to apply our algorithms to the reduced-sized problem. Just as an example, to solve a problem with $n = 48,842$ (the Adult dataset [268]), 5 groups, and $k = 50$, it took one minute on a commodity laptop. More detailed experiments, including runtimes and objective values, can be found in Section 7.6.4.

7.1.1 Approach and Techniques

Our starting point is a LP relaxation of the problem. The integrality gap of the natural LP relaxation is $m$ [74]. For our bicriteria result, we use an iterative rounding procedure, inspired by [269]. In each iteration, we solve an LP whose constraints change from one iteration to the next. We show that the feasible region of the final LP is the intersection of a matroid polytope and $m$ affine spaces. This implies that the size of the support of an optimal extreme solution is at most $k + m$ — see Lemma 7.1.2.1. Rounding up all of these fractional variables results in a solution with $k + m$ centers.

There are two approaches to convert a solution with up to $k + m$ centers to a solution with $k$ centers. The first is to take the best $k$-subset of the $k + m$ centers which results in a $(15 + 6\sqrt{6})^p$-approximation for an additional cost of $O(k^m n(k + m))$ in the running time. This follows from the work of [264]. For completeness, we include it as Lemma 7.2.0.5 in Section 7.2.

The second approach is to “sparsify” the given instance of the problem. We show if the instance is “sparse,” then the integrality gap of the LP is small. A similar idea was
used by [270] for the classic $k$-median problem. We extend this sparsification technique to socially fair clustering. We define an $\alpha$-sparse instance for the socially fair $k$-median problem as an instance in which for an optimum set of facilities $O$, any group $s \in [m]$ and any facility $i$ not in the optimum solution, the number of clients of group $s$ in a ball of radius $d(i,O)/3$ centered at $i$ is less than $\frac{3\alpha|A_s|}{d(i,O)}$. For such an instance, given a set of facilities, replacing facility $i$ with the closest facility to $i$ in $O$ can only increase the total cost of the clients served by this facility by a constant factor plus $2\alpha$. We show that if an instance is $O(\frac{\text{opt}}{m})$-sparse, then the integrality gap of the LP is constant.

For an $O(\frac{\text{opt}}{m})$-sparse instance of the socially fair $k$-median problem, a solution with $k + m$ centers can be converted to a solution with $k$ centers in time $n^{O(m^2)}$ while increasing the objective value only by a constant factor. Our conversion algorithm is based on the fact that there are at most $O(m^2)$ facilities that are far from the facilities in the optimal solution. We enumerate candidates for these facilities and then solve an optimization problem for the facilities that are close to the facilities in the optimal solution. This optimization step is again over the intersection of the polytope of a matroid with $m$ half-spaces.

In summary, our algorithm consists of three main steps.

1. We produce $n^{O(m^2)}$ instances of the problem such that at least one is $O(\frac{\text{opt}}{m})$-sparse and its optimal objective value is equal to that of the original instance (Section 7.3, Lemma 7.3.0.2).

2. For each of the instances produced in the previous step, we compute a pseudo-solution with at most $k+m$ centers using an iterative rounding procedure (Section 7.2, Lemma 7.2.0.1).

3. We convert each pseudo-solution with $k + m$ centers to a solution with $k$ centers (Section 7.4, Lemma 7.4.0.2) and return the solution with the minimum cost.

In addition to constant factor approximation results, we show how the size of the problem can be reduced to $O(km \log n)$ points while preserving the optimality up to a constant
factor. This allows our approach to be scaled to large-size datasets. To achieve this, we combine ideas from the coresets literature, the bicriteria approximation results for the $k$-means++ algorithm, and a result regarding the best solution (with exactly $k$-centers) out of a bicriteria solution. The main difference between our approach and the usual coreset approach is that in addition to the number of clients, we also reduce the number of facilities in the problem to $O(km \log n)$.

7.1.2 Preliminaries

We use terms centers and facilities interchangeably. For a set $S$ and item $i$, we denote $S \cup \{i\}$ by $S + i$. For sets $S_1, \ldots, S_k$, we denote their Cartesian product by $\bigotimes_{j \in [k]} S_j$, i.e., $(s_1, \ldots, s_k) \in \bigotimes_{j \in [k]} S_j$ if and only if $s_1 \in S_1, \ldots, s_k \in S_k$. For an instance $I$ of the problem, we denote an optimal solution of $I$ and its objective value by $\text{OPT}_I$ and $\text{opt}_I$, respectively. A pair $\mathcal{M} = (E, I)$, where $I$ is a non-empty family of subsets of $E$, is a matroid if: 1) for any $S \subseteq T \subseteq E$, if $T \in I$ then $S \in I$ (hereditary property); and 2) for any $S, T \in I$, if $|S| < |T|$, then there exists $i \in T \setminus S$ such that $S + i \in I$ (exchange property) see [271]. We call $I$ the set of independent sets of the matroid $\mathcal{M}$. The basis of $\mathcal{M}$ are all the independent sets of $\mathcal{M}$ of maximal size. The size of all of the basis of a matroid is equal and is called the rank of the matroid. We use the following lemma in the analysis of both our bicriteria algorithm and the algorithm with exactly $k$ centers.

Lemma 7.1.2.1. [194] Let $\mathcal{M} = (E, I)$ be a matroid with rank $k$ and $P(\mathcal{M})$ denote the convex hull of all basis of $\mathcal{M}$. Let $Q$ be the intersection of $P(\mathcal{M})$ with $m$ additional affine constraints. Then any extreme point of $Q$ has a support of size at most $k + m$.

7.1.3 Related Work

Unsupervised learning under fairness constraints has received significant attention over the past decade. Social fairness (i.e., equitable cost for demographic groups) has been considered for problems such as PCA [66, 71]. Other notions of fairness are also considered
Figure 7.1: (a) Distance of $i'$ from the facilities of its representative $i$. (b) Solid and dashed circles are the balls corresponding to representative ($U^*$) and non-representative clients ($U'$), respectively.

for clustering. The most notables are balance in clusters (i.e., equitable representation of groups) [224, 272, 228, 261], balance in representation (i.e., equitable representation of groups in selected centers) [273, 274, 275], and individual fairness [276, 277, 278]. However as shown by [279], different notions of fairness are incompatible in the sense that they cannot be satisfied simultaneously. For example, see the discussion and experimental result regarding incompatibility of social fairness and equitable representation in [72]. In addition, several other notions of fairness for clustering has been considered [249, 280, 281, 282, 283].

Coresets were first studied for the $k$-median and $k$-means problems to improve the running time of $(1 + \epsilon)$-approximation algorithms [267, 284, 285]. This has resulted in algorithms that are only exponential in $k$ and $1/\epsilon$ instead of $n$. They have been used to speed up a variety of fair clustering algorithms as well, including the ones for proportional fairness and individual fairness [245, 248, 227, 286].

$k$-means++ algorithm is an efficient sampling approach for picking the centers for the $k$-means problem [204] and it can also be used for initialization of Lloyd’s heuristic. Such an initialization guarantees an $O(\log k)$-approximation. Selecting more than $k$ centers using the $k$-means++ algorithms results in bicriteria constant factor approximation [287]. This has been used as a seeding method for the random selection of coresets for the $k$-means problem [288]. Recent progress has shown that augmenting the $k$-means++ algorithm with a local search procedure guarantees constant factor approximations [289, 290]. Although the bicriteria approximation is generalized to the $(\ell, p)$-clustering algorithm and general
metrics, the latter is only analyzed for Euclidean $k$-means.

7.2 Bicriteria Approximation

In this section, we prove Theorem 7.1.0.1. Our method relies on solving a series of linear programs and utilizing the iterative rounding framework for the $k$-median problem as developed in [269, 291]. We aim for a cleaner exposition over smaller constants below. We use the following standard linear programming (LP) relaxation (LP1).

\[
\begin{align*}
\min & \quad z \\
\text{s.t.} & \quad z \geq \sum_{i \in \mathcal{A}, j \in \mathcal{F}} w_s(i) d(i, j)^p x_{ij}, \\
& \quad \forall 1 \leq s \leq m, \\
& \quad x_{ij} \leq y_j, \forall i \in \mathcal{A}, j \in \mathcal{F}, \\
& \quad \sum_{j \in \mathcal{F}} y_j = k, \\
& \quad \sum_{j \in \mathcal{F}} x_{ij} = 1, \forall i \in \mathcal{A}, \\
& \quad x, y \geq 0.
\end{align*}
\]

(LP1)

\[
\begin{align*}
\min & \quad z \\
\text{s.t.} & \quad z \geq \sum_{j \in \mathcal{A}, i \in \mathcal{F}_j} w_s(i) d(i, j)^p y_i, \quad (7.2.1)
\end{align*}
\]

(LP2)

\[
\begin{align*}
& \quad \forall 1 \leq s \leq m, \\
& \quad \sum_{j \in \mathcal{F}} y_j = k, \quad (7.2.2) \\
& \quad \sum_{j \in \mathcal{F}_i} y_j = 1, \forall i \in \mathcal{A}, \quad (7.2.3) \\
& \quad y \geq 0.
\end{align*}
\]

Theorem 7.1.0.1 follows as a corollary to Lemma 7.2.0.1 as we can pick all the fractional centers integrally. Observe that, once the centers are fixed, the optimal allocation of clients to facilities is straightforward: every client connects to the nearest opened facility.

**Lemma 7.2.0.1.** Let $0 < \lambda \leq 1$. There is a polynomial time algorithm that given a feasible solution $(\tilde{x}, \tilde{y}, \tilde{z})$ to the linear program LP1 returns a feasible solution $(x', y', z')$ where $z' \leq ((1 + 2(1 + \lambda)/\lambda)(1 + \lambda))^p \tilde{z}$ and the size of the support of $y'$ is at most $k + m$. The running time is polynomial in $n$ and the logarithm of the distance of the farthest points divided by $\lambda$. 

316
Proof. We describe the iterative rounding argument to round the solution \((\bar{x}, \bar{y}, \bar{z})\). As a first step, we work with an equivalent linear program \(\text{LP2}\), where we have removed the assignment variables \(x\). This can be achieved by splitting each facility \(j\) to the number of unique nonzero \(\bar{x}_{ij}\)'s and setting the corresponding variable for these facilities accordingly, e.g., if the unique weights are \(\bar{x}_{1j} < \bar{x}_{2j} < \bar{x}_{3j}\), then the corresponding weights for the facilities are \(\bar{x}_{1j}, \bar{x}_{2j} - \bar{x}_{1j},\) and \(\bar{x}_{3j} - \bar{x}_{2j}\) and the weights of the connections between these new facilities and clients are determined accordingly as either zero or the weight of the facility.

Let \(\overline{F}\) be the set of all (splitted) copies of facilities. Then we can assume \(\bar{x}_{ij} \in \{0, \bar{y}_j\}\) for each \(i, j\) (where \(j \in \overline{F}\)). We set \(F_i = \{j \in \overline{F} : \bar{x}_{ij} > 0\}\). Note that \(F_i\) could contain multiple copies of original facilities. Observe that \(\text{LP2}\) has a feasible solution \((\bar{y}, \bar{z})\) for this choice of \(F_i\) for each \(i \in \mathcal{A}\). Moreover, any feasible solution to \(\text{LP2}\) can be converted to a solution of \(\text{LP1}\) of same cost while ensuring that each client \(i\) gets connected to the original copy of the facilities in \(F_i\).

The iterative argument is based on the following. We group nearby clients and pick only one representative for each group such that if each client is served by the facility that serves its representative, the cost is at most \((1 + 2(1 + \lambda)/\lambda) (1 + \lambda)^p \bar{z}\). Moreover, we ensure that candidate facilities \(F_i\) for representative clients are disjoint. In this case, one observes that the constraints (Equation 7.2.2) and (Equation 7.2.3) in \(\text{LP2}\) define the convex hull of a partition matroid and must be integral. Indeed, this already gives an integral solution to the basic \(k\)-median problem. But, in the socially fair clustering problem, there are \(m\) additional constraints, one for each of the \(m\) groups. Nevertheless, by Lemma 7.1.2.1, any extreme point solution to the matroid polytope intersected with at most \(m\) linear constraints has a support of size at most \(k + m\) (see also [292] Chap. 11).

We now formalize the argument and specify how one iteratively groups the clients. We iteratively remove/change constraints in \(\text{LP2}\) as we do the grouping while ensuring that linear program’s cost does not increase. We initialize \(D_i = \max\{d(i, j) : \bar{x}_{ij} > 0\} = \ldots\)
max\{d(i, j) : j \in F_i\} for each client i. We maintain a set of representative clients \( \mathcal{U}^* \). We say a client \( i \in \mathcal{U}^* \) represents a client \( i' \) if they share a facility, i.e., \( F_i \cap F_{i'} \neq \emptyset \), and \( D_i \leq D_{i'} \). The representative clients do not share any facility with each other. The non-representative clients are put in the set \( \mathcal{U}^f \).

We initialize \( \mathcal{U}^* \) as follows. Sort all clients in increasing order of \( D_i \). Greedily add clients to \( \mathcal{U}^* \) while maintaining \( F_i \cap F_{i'} = \emptyset \) for all \( i \neq i' \in \mathcal{U}^* \). Observe that \( \mathcal{U}^* \) is maximal with above property, i.e., for every \( i' \notin \mathcal{U}^* \), there is \( i \in \mathcal{U}^* \) such that \( F_i \cap F_{i'} \neq \emptyset \) and \( D_i \leq D_{i'} \). We will maintain this invariant in the algorithm. For clients \( i \in \mathcal{U}^f \), we set \( B_i \) to be the facilities in \( F_i \) that are within a distance of \( \frac{D_i}{1+\lambda} \).

In each iteration we solve the following linear program and update \( \mathcal{U}^* \), \( \mathcal{U}^f \), \( D_i \)'s, \( B_i \)'s, and \( F_i \)'s.

\[
\min z \quad \text{(LP(\( \mathcal{U}^*, \mathcal{U}^f, D \))} \\
\text{s.t. } z \geq \sum_{i \in A_i \cap \mathcal{U}^*} \sum_{j \in F_i} d(i, j)^p y_j + \sum_{i \in A_i \cap \mathcal{U}^f} w_i(s) \left( \sum_{j \in B_i} d(i, j)^p y_j + (1 - y(B_i))D_i^p \right), \forall 1 \leq s \leq m, \tag{7.2.4}
\]
\[
\sum_{j \in F_i} y_j = k, \tag{7.2.5}
\]
\[
\sum_{j \in F_i} y_j = 1, \forall i \in \mathcal{U}^*, \tag{7.2.6}
\]
\[
\sum_{j \in B_i} y_j \leq 1, \forall i \in \mathcal{U}^f, \tag{7.2.7}
\]
\[
y \geq 0. \tag{7.2.8}
\]

For clients in \( i \in \mathcal{U}^f \), we only insist that we pick at most one facility from \( B_i \) (see (Equation 7.2.7)). The objective is modified to pay \( D_i \) for any fractional shortfall of facilities in this smaller ball (see (Equation 7.2.4)). Observe that if this additional constraint (Equation 7.2.7) becomes tight for some \( j \in \mathcal{U}^f \), we can decrease \( D_i \) by a factor of \( (1 + \lambda) \) for this client and then update \( \mathcal{U}^* \) accordingly to see if \( i \) can be included in it. Also, we
**Algorithm 24: Iterative Rounding**

1. **Input:** $\mathcal{A} = A_1 \cup \cdots \cup A_m, \mathcal{F}, k, d, \lambda$
2. **Output:** A set of centers of size at most $k + m$.
3. Solve LP$_1$ to get optimal solution $(x^*, y^*, z^*)$ and reate set $\overline{\mathcal{F}}$ by splitting facilities.
4. Set $F_i = \{ j \in \overline{\mathcal{F}} : x^*_{ij} > 0 \}$, $D_i = \max \{ d_{ij} : x^*_{ij} > 0 \}$ for each $i \in \mathcal{U}$.
5. Sort clients in increasing order of $D_i$ and greedily include clients in $\mathcal{U}^*$ while maintaining that $\{ F_i : i \in \mathcal{U}^* \}$ remain disjoint.
6. Set $\mathcal{U}^f = \mathcal{A} \setminus \mathcal{U}^*$.
7. **while** there is some tight constraint from (Equation 7.2.7) **do**
8. **if** there exists $i \in \mathcal{U}^f$ such that $y(B_i) = 1$ (i.e., (Equation 7.2.7) is tight for $i$) **then**
   
   - $F_i \leftarrow B_i, D_i \leftarrow \frac{D_i}{1 + \lambda}, B_i \leftarrow \{ j \in F_i : d(i, j) \leq \frac{D_i}{1 + \lambda} \}$
   - Update-$\mathcal{U}^*_i$.
9. **end if**
10. **end while**
11. Find an extreme point solution $y$ to the linear program LP($\mathcal{U}^*, \mathcal{U}^f, D$).
12. **return** the support of $y$ in the solution of LP($\mathcal{U}^*, \mathcal{U}^f, D$).
13. **Procedure** Update-$\mathcal{U}^*_i$:
14. **if** for every $i' \in \mathcal{U}^*$ that $F_i \cap F_{i'} \neq \emptyset, D_{i'} > D_i$ **then**
15. Remove all $i'$ that represent $i$ from $\mathcal{U}^*$ and add them to $\mathcal{U}^f$.
16. **end if**
17. $\mathcal{U}^* \leftarrow \mathcal{U}^* \cup \{ i \}$.

round each $d(i, j)$ to the nearest power of $(1 + \lambda)$. This only changes the objective by a factor of $(1 + \lambda)^p$ and we abuse notation to assume that $d$ satisfies this constraint (it might no longer be a metric but in the final assignment, we will work with its metric completion).

The iterative algorithm runs as described in Algorithm 24. It is possible that a client moves between $\mathcal{U}^f$ and $\mathcal{U}^*$ but any time that a point is processed in $\mathcal{U}^f$ (Step 3(a) above), $D_i$ is divided by $(1 + \lambda)$. Thus the algorithm takes $O(n \log \frac{\text{diam}}{\lambda})$ iterations, where diam is the distance between the two farthest points. Finally the result is implied by the following claims.

**Claim 7.2.0.2.** The cost of the LP is non-increasing over iterations. Moreover, when the algorithm ends, there are at most $k + m$ facilities in the support.

**Proof.** In each iteration, we put at most one client in $\mathcal{U}^*$. For this client, (Equation 7.2.7)
is tight, i.e., $\sum_{j \in B_i} y_j = 1$. Note that we update $F_i$ to $B_i$. Therefore the new point in $U^*$ satisfies (Equation 7.2.6). Moreover, for a point $i'$ that is removed from $U^*$, we have

$$\sum_{j \in B_{i'}} y_j \leq \sum_{j \in F_{i'}} y_j = 1.$$ 

Therefore such a point satisfies (Equation 7.2.7). Hence a feasible solution to the LP of iteration $t$ is also feasible for iteration $t + 1$. Therefore the cost of the LP is non-increasing over iterations.

The second statement follows since if no constraint from (Equation 7.2.7) is tight, then the linear program is the intersection of a matroid polytope with $m$ linear constraints and the result follows from Lemma 7.1.2.1.

**Claim 7.2.0.3.** For any client $i' \in U^f$, there is always one total facility at a distance of at most $(1 + 2(1 + \lambda)/\lambda) D_{i'}$, i.e., $\sum_{j : d(i', j) \leq (1 + 2(1 + \lambda)/\lambda)D_{i'}} y_j \geq 1$.

**Proof.** Let $t$ be the iteration where $D_{i'}$ is updated for the last time. If $D_{i'}$ is only set once at Line 4 of Algorithm 24 and it is never updated, then $t = 0$. We first show that immediately after iteration $t$, there is one total facility at a distance of at most $3D_{i'}$ from $i'$. If $t = 0$, then there existed $i \in U^*$ such that $F_i \cap F_{i'} \neq \emptyset$ and $D_i \leq D_{i'}$. Therefore by triangle inequality, all the facilities in $F_i$ are within a distance of at most $3D_{i'}$ from $i'$, see Fig. 7.1 (a). Hence because (Equation 7.2.6) enforces one total facility in $F_i$, there exists one total facility at a distance of at most $3D_{i'}$ from $i'$. If $t > 0$, then $i'$ is moved from $U^*$ to $U^f$ because at iteration $t$, a facility $i$ is added to $U^*$ such that $D_i < D_{i'}$ and $F_i \cap F_{i'} \neq \emptyset$ — see the condition of Procedure Update-$U^*$($i$) in Algorithm 24. Again because of enforcement of (Equation 7.2.6) and triangle inequality, there exists one total facility at a distance of at most $3D_{i'}$ from $i'$ immediately after iteration $t$.

Now note that after iteration $t$, the facility $i \in U^*$ with $F_i \cap F_{i'} \neq \emptyset$ and $D_i \leq D_{i'}$ might get removed from $U^*$. In which case, we do not have the guarantee of (Equation 7.2.6) any longer. Let $i_0 := i$. We define $i_{p+1}$ to be the client that has caused the removal of client $i_p$.
(through Procedure Update-$\mathcal{U}^*(i)$) from $\mathcal{U}^*$ after iteration $t$. Note that by the condition of Update-$\mathcal{U}^*(i)$ from $\mathcal{U}^*$, $D_{i_{p+1}} < D_{i_p}$. Therefore because we have rounded the distances to multiples of $(1 + \lambda)$, we have $D_{i_{p+1}} \leq \frac{D_{i_p}}{1+\lambda}$. Let $i_r$ be the last point in this chain, i.e., $i_r$ has caused the removal of $i_{r-1}$ and $i_r$ has stayed in $\mathcal{U}^*$ until termination of the algorithm. Then by guarantee of (Equation 7.2.6) and triangle inequality, there is one total facility for $i'$ within a distance of

$$D_{i'} + \sum_{j=0}^{r} 2D_{i_j} \leq D_{i'} + 2 \sum_{j=0}^{r} \frac{D_{i'_j}}{(1+\lambda)^j} \leq \left(1 + \frac{2(1+\lambda)}{\lambda}\right) D_{i'}.$$


Claim 7.2.0.4. Let $\hat{y}$ be an integral solution to linear program $LP(\mathcal{U}^*, \mathcal{U}^f, D)$ after the last iteration. Then, there is a solution $(\hat{x}, \hat{y})$ to the linear program $LP_1$ such that objective is at most

$$(1 + 2(1+\lambda)/\lambda) \left(1 + \lambda \right)^p$$

times the objective of the linear program $LP(\mathcal{U}^*, \mathcal{U}^f, D)$.

Proof. By Claim 7.2.0.2, at every iteration, the cost of the linear program only decreases since a feasible solution to previous iteration remains feasible for the next iteration. Thus the objective value of $\hat{y}$ in $LP(\mathcal{U}^*, \mathcal{U}^f, D)$ is at most $(1+\lambda)^p$ the optimal cost of $LP_1$ (where we lost the factor of $(1 + \lambda)^p$ by rounding all distances to powers of $(1 + \lambda)$).

We now construct $\hat{x}$ such that $(\hat{x}, \hat{y})$ is feasible to $LP_1$. First note that the above procedure always terminates. We construct $\hat{x}$ by processing clients one by one. We process the clients in $\mathcal{U}^f$ and $\mathcal{U}^*$ as follows. For any $i \in \mathcal{U}^*$, we define $\hat{x}_{ij} = \hat{y}_j$ for each $j \in F_i$. Observe that we have $\sum_{j \in F_i} x_{ij} = 1$ for such $i \in \mathcal{U}^*$ and we obtain feasibility for this client. For any $i \in \mathcal{U}^f$, we define $\hat{x}_{ij} = \hat{y}_j$ for each $j \in B_i$. Observe that we only insisted $\sum_{j \in B_i} \hat{y}_j \leq 1$ and therefore we still need to find $1 - \sum_{j \in B_i} \hat{x}_{ij} = 1 - \sum_{j \in B_i} \hat{y}_j$ facilities to assign to client $i$. For this remaining amount $1 - \hat{y}(B_i)$, we notice by Claim 7.2.0.3, there

321
is at least one facility within distance \( \left( 1 + \frac{2(1+\lambda)}{\lambda} \right) D_i \) of this client. Thus we can assign the remaining \( 1 - \hat{\gamma}(B_i) \) facility to client \( i \) at a distance of no more than \( \left( 1 + \frac{2(1+\lambda)}{\lambda} \right) D_i \). Note that the cost is only increased by a factor of \( \left( 1 + \frac{2(1+\lambda)}{\lambda} \right)^p \).

Now we prove Theorem 7.1.0.1 by substituting the best \( \lambda \) in Lemma 7.2.0.1.

\textit{Proof of Theorem 7.1.0.1.} By Lemma 7.2.0.1, the output vector of Algorithm 24 corresponding to the centers has a support of size at most \( k + m \). Rounding up all the fractional centers, we get a solution with at most \( k + m \) centers and a cost of at most

\[
\left( \left( 1 + \frac{2(1+\lambda)}{\lambda} \right) \right)^p
\]

of the optimal. We optimize over \( \lambda \) by taking the gradient of \( \left( 1 + \frac{2(1+\lambda)}{\lambda} \right) \) and setting it to zero. This gives the optimum value of \( \lambda = \sqrt{2/3} \). Substituting this, gives a total approximation factor of \( (5 + 2\sqrt{6})^p \). \( \square \)

\textbf{Lemma 7.2.0.5 \cite{264}.} Let \( k' > k \) and \( S \) be a set of centers of size \( k' \) and cost \( C \) for the socially fair \((\ell_p, k)\)-clustering problem with \( m \) groups. Let \( T \subset S \) be a set of size \( k \) with minimum cost among all subsets of size \( k \) of \( S \). Then the cost of \( T \) is less than or equal to \( 3^{p-1}(C + 2\text{opt}) \) where \( \text{opt} \) is the cost of the optimal solution.

\textit{Proof.} Let \( \text{OPT} \) be an optimal set of centers. For each center \( o \in \text{OPT} \), let \( s_o \) be the closest center in \( S \) to \( o \), i.e., \( s_o := \arg \min_{s \in S} d(s, o) \). Let \( T' := \{ s_o : o \in \text{OPT} \} \). Because the size of \( \text{OPT} \) is \( k \), \( |T'| \leq k \). We show that the cost of \( T' \) is less than or equal to \( 3^{p-1}(C + 2\text{opt}) \).

The result follows from this because \( T' \subset S \) and \( |T'| \leq k \).

Let \( i \) be a client and \( o_i \) be the closest facility in \( \text{OPT} \) to \( i \). Let \( t'_i \) be the closest facility to \( o_i \) in \( T' \) which means \( t'_i \) is also the closest facility to \( o_i \) in \( S \). Moreover let \( s_i \) be the closest facility to \( i \) in \( S \). By triangle inequality, \( d(i, t'_i) \leq d(i, o_i) + d(o_i, t'_i) \). By definition of \( t'_i \), \( d(o_i, t'_i) \leq d(o_i, s_i) \). Therefore \( d(i, t'_i) \leq d(i, o_i) + d(o_i, s_i) \). Moreover by triangle
Algorithm 25: Sparsify

\begin{algorithm}
\caption{Sparsify}
\begin{algorithmic}
\Statex \textbf{Input:} $A = A_1 \cup \cdots \cup A_m, \mathcal{F}, k, d, t \in \mathbb{N}$
\Statex \textbf{Output:} A set of fair $k$-median instances.
\For {$t' = 1, \ldots, m^2 t$ and $t'$ facility pairs $(j_1, j'_1), \ldots, (j_r, j'_r)$}
\Statex Output $I' = (\mathcal{F}', A, k, d)$, where $\mathcal{F}' = \mathcal{F} \setminus \bigcup_{r=1}^t \text{FBALL}(j_r, d(j_r, j'_r))$.
\EndFor
\end{algorithmic}
\end{algorithm}

inequality $d(o_i, s_i) \leq d(i, o_i) + d(i, s_i)$. Therefore $d(i, t'_i) \leq 2d(i, o_i) + d(i, s_i)$. Taking both sides to the power of $p$ and using the power mean inequality, i.e., $(x + y + z)^p \leq 3^{p-1}(x^p + y^p + z^p)$, we conclude $d(i, t'_i)^p \leq 3^{p-1}(2d(i, o_i)^p + d(i, s_i)^p)$. The result follows from summing such inequality for each group and taking the maximum over groups. \hfill $\square$

### 7.3 Approximation Algorithms for Fair $k$-Clustering

We first show how to generate a set of instances such that at least one of them is sparse and has the same optimal objective value as the original instance. Then we present our algorithm to find a solution with $k$ facilities from a pseudo-solution with $k + m$ facilities for a sparse instance, inspired by [270]. We need to address new difficulties: the sparsity with respect to all groups $s \in [m]$; and as our pseudo-solution has $m$ additional centers (instead of $O(1)$ additional centers), we need a sparser instance compared to [270].

One new technique is solving the optimization problem given in Step 11 of Algorithm 26 (Lemma 7.4.0.1). This is trivial for the vanilla $k$-median but in the fair setting, we use certain properties of the extreme points of intersection of a matroid polytope with half-spaces, and combine this with a careful enumeration.

For an instance $I$, we denote the cost of a set of facilities $F$ by $\text{cost}_I(F)$. For a point $q$ and $r > 0$, we denote the facilities in the ball of radius $r$ at $q$ by $\text{FBall}_I(q, r)$. This does not contain facilities at distance exactly $r$ from $q$. For a group $s \in [m]$, the set of clients of $s$ in the ball of radius $r$ at $q$ is $\text{CBall}_{I,s}(q, r)$. Note that because we consider the clients as weights on points, $\text{CBall}_{I,s}(q, r)$ is actually a set of (point, weight) pairs. We let $|\text{CBall}_{I,s}(q, r)| = \sum_{i \in \text{CBall}_{I,s}(q, r)} w_s(i)$. 

323
Definition 7.3.0.1. [Sparse Instance] For $\alpha > 0$, an instance of the fair $\ell_p$ clustering problem $I = (k, F, A, d)$ is $\alpha$-sparse if for each facility $j \in F$ and group $s \in [m],$

$$\left( \frac{2}{3} d(j, \text{OPT}_I) \right)^p \cdot |C\text{Ball}_{I,s}(j, \frac{1}{3} d(j, \text{OPT}_I))| \leq \alpha.$$ We say that a facility $j$ is $\alpha$-dense if it violates the above for some group $s \in [m]$.

To motivate the definition, let $I$ be an $\alpha$-sparse instance, $\text{OPT}_I$ be an optimal solution of $I$, $j$ be a facility not in $\text{OPT}_I$ and $j^*$ be the closest facility in $\text{OPT}_I$ to $j$. Let $F$ be a solution that contains $j$ and $\eta_{j,s}$ be the total cost of the clients of group $s \in [m]$ that are connected to $j$ in solution $F$. Then,

$$(\text{cost of group } s \text{ for solution } F \cup j \setminus j^*)$$

$$\leq (\text{cost of group } s \text{ for solution } F) + 2^O(p) \cdot (\alpha + \eta_{j,s}).$$

This property implies that if $\alpha \leq \frac{\text{opt}_I}{m}$, then replacing $m$ different facilities can increase the cost by a factor of $2^O(p)$ plus $2^O(p) \cdot \text{opt}_I$, and the integrality gap of the LP relaxation is $2^O(p)$. The next algorithm generates a set of instances such that at least one of them has objective value equal to $\text{opt}_I$ and is $(\text{opt}_I/mt)$-sparse for a fixed integer $t$.

Lemma 7.3.0.2. Algorithm 25 runs in $n^{O(m^2 t)}$ time and produces instances of the socially fair $\ell_p$ clustering problem such that at least one of them satisfies the following: (1) The optimal value of the original instance $I$ is equal to the optimal value of the produced instance $I'$; (2) $I'$ is $\frac{\text{opt}_I}{mt}$-sparse.

Proof. First note that a facility $i$ in $\text{OPT}_I$ cannot be $\alpha$-dense because $d(i, \text{OPT}_I)) = 0$. Let $(j_1, j'_1), \ldots, (j_t, j'_t)$ be a sequence of pairs of facilities such that for every $b = 1, \ldots, \ell,$

- $j_b \in F \setminus \bigcup_{z=1}^{b-1} F\text{Ball}_{I}(j_z, d(j_z, j'_z))$ is an $\frac{\text{opt}_I}{mt}$-dense facility; and
- $j'_b$ is the closest facility to $j_b$ in $\text{OPT}_I.$
We show that $\ell \leq m^2t$. For $b \in [\ell]$ and $s \in [m]$, let $\mathcal{B}_{b,s} := \text{CBall}_{I,s}(j_b, \frac{1}{3}d(j_b, j'_b))$. First we show that for any group $s \in [m]$, the client balls $\mathcal{B}_{1,s}, \ldots, \mathcal{B}_{\ell,s}$ are disjoint. Let $1 \leq z < w \leq \ell$. By triangle inequality $d(j_w, j'_z) \leq d(j_w, j_z) + d(j_z, j'_z)$. Moreover by definition $j_w \notin \text{FBall}_{I}(j_z, d(j_z, j'_z))$. Thus $d(j_z, j'_z) \leq d(j_w, j_z)$. Hence $d(j_w, j'_z) \leq 2d(j_w, j_z)$. Since $j'_w$ is the closest facility to $j_w$ in $\text{OPT}_I$, $d(j_w, j'_w) \leq d(j_w, j'_z)$. Therefore $d(j_w, j'_w) \leq 2d(j_w, j_z)$. Combining this with $d(j_z, j'_z) \leq d(j_w, j_z)$ implies $\frac{1}{3}(d(j_z, j'_z) + d(j_w, j'_w)) \leq d(j_z, j_z)$. If $\mathcal{B}_{z,s}$ and $\mathcal{B}_{w,s}$ overlap then there exists $u \in \mathcal{B}_{z,s} \cap \mathcal{B}_{w,s}$ and by triangle inequality, $d(j_z, j_w) \leq d(j_z, u) + d(j_w, u) < \frac{1}{3}d(j_z, j'_z) + \frac{1}{3}d(j_w, j'_w)$, which is a contradiction.

Therefore for $s \in [m]$, $\mathcal{B}_{1,s}, \ldots, \mathcal{B}_{\ell,s}$ are disjoint. Also since $A_1, \ldots, A_s$ are disjoint, all of $\mathcal{B}_{b,s}$'s are disjoint for $b \in [\ell]$ and $s \in [m]$. By definition, for any $b \in [\ell]$, there exists $s_b \in [m]$ such that $\left(\frac{2}{3}d(j_b, \text{OPT}_I)\right)^p |\mathcal{B}_{b,s_b}| > \frac{\text{OPT}_I}{m^2t}$. Therefore, if $\ell > m^2t$, $\sum_{b=1}^{\ell} \left(\frac{2}{3}d(j_b, \text{OPT}_I)\right)^p |\mathcal{B}_{b,s_b}| > m \text{OPT}_I$. Thus

$$m \cdot \max_{s \in [m]} \sum_{b=1}^{\ell} \left(\frac{2}{3}d(j_b, \text{OPT}_I)\right)^p |\mathcal{B}_{b,s}| \geq \sum_{s \in [m]} \sum_{b=1}^{\ell} \left(\frac{2}{3}d(j_b, \text{OPT}_I)\right)^p |\mathcal{B}_{b,s}| > m \text{OPT}_I.$$  

Note that the connection cost of a client in $\mathcal{B}_{b,s}$ in the optimal solution is at least $\left(\frac{2}{3}d(j_b, \text{OPT}_I)\right)^p = \left(\frac{2}{3}d(j_b, j'_b)\right)^p$. Therefore, as the $\mathcal{B}_{b,s}$'s are disjoint,

$$\text{OPT}_I \geq \max_{s \in [m]} \sum_{b=1}^{\ell} \left(\frac{2}{3}d(j_b, \text{OPT}_I)\right)^p |\mathcal{B}_{b,s}|.$$  

This is a contradiction. Therefore $\ell \leq m^2t$. Thus Algorithm 25 returns an instance with the desired properties. \hfill $\Box$

### 7.4 Converting a Solution with $k + m$ centers to one with $k$ centers

We first analyze the special case when the set of facilities is partitioned to $k$ disjoint sets and we are constrained to pick exactly one from each set. This will be a subroutine in our
Lemma 7.4.0.1. Let $S_1, \ldots, S_k$ be disjoint sets such that $S_1 \cup \cdots \cup S_k = [n]$. For $g \in [m]$, $j \in [k]$, $v \in S_j$, let $\alpha_v^{(g,j)} \geq 0$. Then there is an $(nk)^{O(m^2/\epsilon)}$-time algorithm that finds a $(1 + \epsilon)$-approximate solution to $\min_{v_i \in S_i; i \in [k]} \max_{g \in [m]} \sum_{j \in [k]} \alpha_v^{(g,j)}$.

Proof. The LP relaxation of the above problem is

$$\min \theta \quad \text{such that} \quad \sum_{j \in [k]} \sum_{v \in S_j} \alpha_v^{(g,j)} x_v^{(j)} \leq \theta, \forall g \in [m],$$

$$\sum_{v \in S_j} x_v^{(j)} = 1, \forall j \in [k],$$

$$x^{(j)} \geq 0, \forall j \in [k].$$

Note that this is equivalent to optimizing over a partition matroid with $m$ extra linear constraints. Therefore by Lemma 7.1.2.1, an extreme point solution has a support of size at most $k + m$. Now suppose $\theta^*$ is the optimal integral objective value, and $v_1^* \in S_1, \ldots, v_k^* \in S_k$ are the points that achieve this optimal objective. For each $g \in [m]$, at most $m/\epsilon$ many $\alpha_v^{(g,j)}$, $j \in [k]$, can be more than $\frac{\epsilon}{m} \theta^*$ because $\sum_{j \in [k]} \alpha_v^{(g,j)} \leq \theta^*$. Suppose, for each $g \in [m]$, we guess the set of indices

$$T_g = \{ j \in [k] : \alpha_v^{(g,j)} \geq \frac{\epsilon}{m} \theta^* \}.$$

This takes $k^{O(m^2/\epsilon)}$ time by enumerating over set $[k]$. Let $T = T_1 \cup \cdots \cup T_m$. For $j \in T$, we also guess $v_j^*$ in the optimum solution by enumerating over $S_j$'s, $j \in T$. This increases the running time by a multiplicative factor of $n^{O(m^2/\epsilon)}$ since $v_j^* \in S_j$ and $|S_j| \leq n$. Based on these guesses, we can set the corresponding variables in the LP, i.e., for each $S_j$ such that $j \in T$, we add the following constraints for $v \in S_j$. $x_v^{(j)} = 1$ if $v = v_j^*$, and $x_v^{(j)} = 0$, otherwise. The number of LPs generated by this enumeration is $(nk)^{O(m^2/\epsilon)}$.

We solve all such LPs to get optimum extreme points. Let $(\bar{\theta}, \bar{x}^{(1)}, \ldots, \bar{x}^{(k)})$ be an
optimum extreme point for the LP corresponding to the right guess (i.e., the guess in which we have identified all indices \( j \in [k] \) along with their corresponding \( v_j^* \) such that there exists \( g \in [m] \) where \( \alpha_{v_j^*}^{(g,j)} \geq \frac{\varepsilon}{m} \theta^* \)). Therefore \( \overline{\theta} \leq \theta^* \). Let

\[
R = \{ j \in [k] : x(j) \not\in \{0, 1\}^{[S_j]} \}.
\]

Since the feasible region of the LP corresponds to the intersection of a matroid polytope and \( m \) half-spaces, by Lemma 7.1.2.1, the size of the support of an extreme solution is \( k + m \). Moreover any cluster with \( j \in [k] \) that have fractional centers in the extreme point solution, contributes at least 2 to the size of the support because of the equality constraint in the LP. Therefore \( 2|R| + (k - |R|) \leq k + m \) which implies \( |R| \leq m \).

Now we guess the \( v_j^* \) for all \( j \in R \). By construction, \( R \cap T = \emptyset \). Therefore for all \( j \in R \) and \( g \in [m] \), \( \alpha_{v_j^*}^{(g,j)} < \frac{\varepsilon}{m} \theta^* \). Therefore for all \( g \in [m] \),

\[
\sum_{j \in R} \alpha_{v_j^*}^{(g,j)} \leq m \cdot \frac{\varepsilon}{m} \theta^* = \varepsilon \theta^*.
\]

Thus for the right guess of \( v_j^* \), \( j \in R \), we get an integral solution with a cost less than or equal to \( \overline{\theta} + \varepsilon \theta^* \leq (1 + \varepsilon) \theta^* \). \qed

Algorithm 26 is our main procedure to convert a solution with \( k+m \) centers to a solution with \( k \) centers. We need \( \beta \) to be in the interval mentioned in Lemma 7.4.0.2. To achieve this we guess \( \text{opt}_I \) as different powers of two and try the corresponding \( \beta \)'s. The main idea behind the algorithm is that in a pseudo-solution of a sparse instance, there are only a few (\(< m^2 t\)) facilities that are far from facilities in the optimal solution. So the algorithm tries to guess those facilities and replace them by facilities in the optimal solution. For the rest of the facilities in the pseudo-solution (which are close to facilities in the optimal solution), the algorithm solves an optimization problem (Lemma 7.4.0.1) to find a set of facilities with a cost comparable to the optimal solution.
Algorithm 26: Obtaining a solution from a pseudo-solution

1. **Input:** Instance $I$, $\beta$, a pseudo-solution $T$, $\epsilon' > 0$, $\delta \in (0, \min\{\frac{1}{8}, \frac{\log(1+\epsilon')}{12}\})$, and integer $t \geq 4 \cdot (1 + \frac{3}{\delta})^p$.

2. **Output:** A solution with at most $k$ centers.

3. $T' \leftarrow T$

4. while $|T'| > k$ and there is $j \in T'$ s.t. $\text{cost}_I(T' \setminus \{j\}) \leq \text{cost}_I(T') + \beta$

5. $T' \leftarrow T' \setminus \{j\}$

6. if $|T'| \leq k$ then return $T'$;

7. for all $D \subseteq T'$ and $V \subseteq F$ such that $|D| + |V| = k$ and $|V| < m^2 \cdot t$

8. For $s \in [m]$, $j \in D$, $f_j \in \text{FBall}_I(j, \delta L_j)$, set $\alpha^{(s,j)}_{f_j} = \sum_{i \in \text{CBall}_{I,s}(j, L_j/3)} \min\{d(i, f_j)^p, d(i, V)^p\}$.

9. Let $(\tilde{f}_j : j \in D) \in \bigotimes_{j \in D} \text{FBall}_I(j, \delta L_j)$ be $(1 + \epsilon)$-approximate solution to (see Lemma 7.4.0.1)

10. return $S := \arg\min_{S \subseteq D, V} \text{cost}_I(S, V)$

Finally combining the following lemma with Lemma 7.3.0.2 (sparsification) and Theorem 7.1.0.1 (bicriteria algorithm) implies Theorem 7.1.0.2.

**Lemma 7.4.0.2.** Let $I = (k, F, A, d)$ be an $\frac{\text{opt}_I}{m^2}$-sparse instance of the $(\ell_p, k)$-clustering problem, $T$ be a pseudo-solution with at most $k + m$ centers, $\epsilon' > 0$,

$$\delta \in (0, \min\{\frac{1}{8}, \frac{\log(1+\epsilon')}{12}\}),$$

$$t \geq 4(1 + \frac{3}{\delta})^p$$

be an integer, and

$$\frac{2}{mt} \left(\text{opt}_I + (1 + \frac{3}{\delta})^p \cdot \text{cost}_I(T)\right) \leq \beta \leq \frac{2}{mt} \left(2 \cdot \text{opt}_I + (1 + \frac{3}{\delta})^p \cdot \text{cost}_I(T)\right).$$

Then Algorithm 26 finds a set $S \in F$ in time $n^{m^2 \cdot 2^{O(p)}}$ such that $|S| \leq k$ and

$$\text{cost}_I(S) \leq (O(1) + (1 + \epsilon')^p) \cdot \left(\text{cost}_I(T) + \text{opt}_I\right)$$

328
Proof. If Algorithm 26 ends in Step 7, then \( \text{cost}_I(T) + m \beta \) is at most \( \text{cost}_I(T) + \frac{2}{3} (2 \text{opt}_I + (1 + \frac{3}{8}) \text{cost}_I(T)) = O(\text{opt}_I + \text{cost}_I(T)) \). Otherwise, we run the loop. Now we show that there exist sets \( D_0 \subseteq T' \) and \( V_0 \subseteq \mathcal{F} \) such that \( |V_0| < m^2 t, |D_0| + |V_0| = k \), and \( S_{D_0} \cdot V_0 \) satisfies the desired properties. For a facility \( j \in T' \), let \( L_j = d(j, T' \setminus \{j\}) \) and \( \ell_j = d(j, \text{OPT}_I) \). We say \( j \in I \) is determined if \( \ell_j \leq \delta L_j \). Otherwise, we say \( j \) is undetermined. Let \( D_0 = \{ j \in T' : \ell_j \leq \delta L_j \} \). For \( j \in D_0 \), let \( f^*_j \) be the closest facility to \( j \) in \( \text{OPT}_I \). Let \( V_0 = \text{OPT}_I \setminus \{ f^*_j : j \in D_0 \} \). First note that for any two distinct facilities in \( j, j' \in D_0 \), \( d(j, j') \geq \max \{L_j, L_{j'}\} \). Moreover by definition, \( d(j, f^*_j) \leq \delta L_j \leq \delta \max \{L_j, L_{j'}\} \). Therefore by triangle inequality, \( d(j', f^*_j) \geq (1 - \delta) \max \{L_j, L_{j'}\} \). Moreover by definition and because \( \delta \in (0, \frac{1}{8}) \), \( (1 - \delta) \max \{L_j, L_{j'}\} > \delta L_j \geq d(j', f^*_j) \). Therefore \( d(j', f^*_j) > d(j', f^*_j) \). Thus for any two distinct \( j, j' \in D_0 \), \( f^*_j \neq f^*_j \).

Therefore \(|\{f^*_j : j \in D_0\}| = |D_0| \). Thus \(|V_0| = |\text{OPT}_I| - |D_0| = k - |D_0| \). Let \( U_0 = T' \setminus D_0 \) be the set of undetermined facilities. Since \(|T'| > k \), \(|V_0| = k - |D_0| = k - |T'| + |U_0| < |U_0| \). We show \(|U_0| < m^2 t \). For every \( j \in T' \) and \( s \in [m] \), let \( A_{s,j} \) be the set of clients of group \( s \) that are connected to \( j \) in solution \( T' \) and let \( C_{s,j} \) be the total connection cost of these clients. Therefore \( \text{cost}_I(T') = \max_{s \in [m]} \sum_{j \in T'} C_{s,j}. \) Let \( j^* := \arg \min_{j \in U_0} \sum_{s \in [m]} C_{s,j}. \) Let \( \tilde{j} \) be the closest facility to \( j^* \) in \( T' \setminus \{j^*\} \), i.e., \( d(j^*, \tilde{j}) = L_{j^*}. \) Then \( \text{cost}_I(T' \setminus \{j^*\}) - \text{cost}_I(T') \leq \max_{s \in [m]} \sum_{i \in A_{s,j^*}} d(i, \tilde{j})^p \). For \( s \in [m] \), let \( A_{s,j^*}^\text{in} := A_{s,j^*} \cap \text{CBall}_{s,j^*}(j^*, \frac{1}{3} \delta L_{j^*}) \) and \( A_{s,j^*}^\text{out} := A_{s,j^*} \setminus A_{s,j^*}^\text{in}. \) By triangle inequality, for any \( i \in A_{s,j^*}^\text{in}, d(i, \tilde{j}) \leq (1 + \frac{1}{3} \delta) L_{j^*}. \) Moreover since \( j^* \) is undetermined, \( d(i, \tilde{j}) < (1 + \frac{1}{3} \delta)^\frac{1}{p} \ell_{j^*} = (\frac{1}{3} + \frac{1}{3}) \ell_{j^*} < \frac{2}{3} \ell_{j^*}, \) and for any \( s \in [m] \), \( \text{CBall}_{s,j^*}(j^*, \frac{1}{3} \delta L_{j^*}) \subseteq \text{CBall}_{s,j^*}(j^*, \frac{1}{3} \ell_{j^*}) \). Thus

\[
\sum_{i \in A_{s,j^*}^\text{in}} d(i, \tilde{j})^p < \left(\frac{2}{3} \ell_{j^*}\right)^p |\text{CBall}_{s,j^*}(j^*, \frac{1}{3} \ell_{j^*})|.
\]

Therefore since \( I \) is a \( \frac{\text{opt}_I}{mt} \)-sparse instance, \( \sum_{i \in A_{s,j^*}^\text{in}} d(i, \tilde{j})^p \leq \frac{\text{opt}_I}{mt}. \) For \( i \in A_{s,j^*}^\text{out}, d(i, j^*) \geq \frac{1}{3} \delta L_{j^*}. \) Thus \( \frac{3}{2} d(i, j^*) \geq L_{j^*} \) and by triangle inequality, \( d(i, \tilde{j}) \leq d(i, j^*) + \frac{3}{2} \delta L_{j^*} \).
\[ d(j^*, \tilde{j}) = d(i, j^*) + L_{j^*} \leq (1 + \frac{3}{\delta})d(i, j^*). \]

Therefore

\[ \text{cost}_I(T' \setminus \{j^*\}) - \text{cost}_I(T') \leq \max_{s \in \{m\}} \left( \frac{\text{opt}_I(T')}{mt} + (1 + \frac{3}{\delta})^p C_{s, j^*} \right) \]

\[ \leq \frac{\text{opt}_I}{mt} + (1 + \frac{3}{\delta})^p \sum_{s \in \{m\}} C_{s, j^*}. \]

(7.4.1)

By definition, \[\sum_{s \in \{m\}} C_{s, j^*} = \min_{j \in U_0} \sum_{s \in \{m\}} C_{s, j} \leq \frac{m \text{cost}_I(T')}{|U_0|}.\] So if \(|U_0| \geq m^2t\), then

\[\sum_{s \in \{m\}} C_{s, j^*} \leq \frac{\text{cost}_I(T')}{mt}.\]

Moreover, since \(|T \setminus T'| < m\),

\[\text{cost}_I(T') < \text{cost}_I(T) + m\beta \leq \text{cost}_I(T) + \frac{2}{t} \left( 2 \text{opt}_I + (1 + \frac{3}{\delta})^p \text{cost}_I(T) \right) \]

\[ \leq \left( \frac{1}{1 + \frac{3}{\delta}} \right)^p \text{opt}_I + \frac{3}{2} \text{cost}_I(T). \]

Combining with (Equation 7.4.1),

\[\text{cost}_I(T' \setminus \{j^*\}) - \text{cost}_I(T') \leq \frac{2 \text{opt}_I}{mt} + \frac{3}{2}(1 + \frac{3}{\delta})^p \frac{\text{cost}_I(T)}{mt} \leq \beta.\]

This is a contradiction because \(j^*\) should be removed in Step 4 of Algorithm 26. Therefore \(|U_0| < m^2t\).

Now, we need to bound the cost of \(S_{D_0, V_0}\). For \(j \in D_0\) and \(s \in [m]\), let

\[i \in \text{CBall}_{I,s}(j, \frac{1}{3}L_j).\]

By triangle inequality the distance of \(i\) to any facility in \(F_{\text{Ball}}(j, \delta L_j)\) is at most \((\frac{1}{3} + \delta)\).

For a facility \(j' \in D_0\), \(j' \neq j\), by triangle inequality and because \(d(j, j') \geq \max\{L_j, L_{j'}\}\), the distance of \(i\) to any facility in \(F_{\text{Ball}}(j', \delta L_{j'})\) is at least

\[d(j, j') - \frac{L_j}{3} - \delta L_{j'} \geq d(j, j') - (\frac{1}{3} + \delta)d(j, j') = (\frac{2}{3} - \delta)d(j, j') \geq (\frac{2}{3} - \delta)L_j.\]

For \(\delta < \frac{1}{8}\), we have \(\frac{1}{3} + \delta < \frac{2}{3} - \delta\). Therefore, \(i\) is either connected to \(f_j\) or to a facility in \(V_0\). Let \(a_{f_j}^{(s,j)}\)'s be as defined in Algorithm 26 for \(D_0\) and \(V_0\). Let \((\tilde{f}_j : j \in D_0)\) be a \((1 + \epsilon)\)-approximate solution for the following, obtained by Lemma 7.4.0.1. For \(s \in [m]\),
let $T_s = \bigcup_{j \in D_0} CBall_{I,s}(j, L_j/3)$. Since for $j \in D_0$, $f_j^* \in \text{OPT}_I$'s are also in balls $FBall_I(j, \delta L_j)$,
\[
\max_{s \in [m]} \sum_{i \in T_s} d(i, S_{D_0,V_0})^p \leq (1 + \epsilon) \max_{s \in [m]} \sum_{i \in T_s} d(i, \text{OPT}_I)^p.
\]

Now consider a client $i \in A_s \setminus T_s$. If in the optimal solution, $i$ is connected to a facility in $V_0$, then by definition, $d(i, \text{OPT}_I) \geq d(i, S_{D_0,V_0})$. Otherwise, in the optimal solution, $i$ is connected to $f_j^* \in FBall_I(j, \delta L_j)$ for some $j \in D_0$. We compare $d(i, \tilde{f}_j)$ to $d(i, f_j^*)$. Since $\tilde{f}_j, f_j^* \in FBall_I(j, \delta L_j)$, by triangle inequality and because $d(i, j) \geq L_j/3$,
\[
\frac{d(i, \tilde{f}_j)^p}{d(i, f_j^*)^p} \leq \frac{(d(i, j) + \delta L_j)^p}{(d(i, j) - \delta L_j)^p} \leq \left(\frac{L_j/3 + \delta L_j}{L_j/3 - \delta L_j}\right)^p = \left(\frac{1 + 3\delta}{1 - 3\delta}\right)^p.
\]

Thus because $\delta \leq \frac{1}{8}$, $\frac{1 + 3\delta}{1 - 3\delta} \leq 1 + 12\delta$. Moreover since $\delta < \frac{\log(1 + \epsilon')}{12}$, $\text{cost}_I(S_{D_0,V_0}) \leq (1 + \epsilon')^p \cdot \text{opt}_I$. Finally note that the loop runs for $n^O(m^2t)$ iterations because $|V| < m^2t$ and $|T \setminus D| \leq m^2t + m$. Moreover by Lemma 7.4.0.1, each iteration runs in $(nk)^O(m^2/\epsilon)$ time. \hfill $\square$

### 7.5 Speeding Up the Algorithm

Our experiments show that solving the LP relaxation of the problem is the main bottleneck in our algorithm. This process can be slow when working with datasets that have tens of thousands of samples. Coresets, which are weighted subsets of the data that approximate the cost of clustering, have been widely used to speed up clustering algorithms [288]. They have also been used for fair clusterings, such as proportional fairness [224] and individual fairness [293], as well as socially fair $(\ell_p, k)$-clustering [294]. Although coresets can improve the running time, it is important to note that they only reduce the number of clients, not facilities. In this section, we demonstrate that combining coresets with the $k$-means++ algorithm and exhaustive search can also reduce the number of facilities. Our main result is as follows.

**Theorem 7.5.0.1.** Let $A := A_1 \cup \cdots \cup A_m$ with weights $w_{A_s}$ for $s \in [m]$ be a set of clients in a metric space and $F := A$ be the set of centers. Let $|A| = n$. There exists an algorithm...
running in time \( O(m \cdot k \cdot n \log n) \) that with high probably outputs \( \tilde{A}_s \subseteq A_s \) with weights \( \tilde{w}_{\tilde{A}_s} \) for all \( s \in [m] \), and \( F \subseteq \mathcal{F} \) such that \( |\tilde{A}_s| = O(k \log n) \) and \( |F| \leq 2mk \) and if \( \tilde{G} \) is a subset of size \( k \) of \( F \) that

\[
\max_{s \in [m]} c_p(\tilde{A}_s, \tilde{w}_{\tilde{A}_s}, \tilde{G}) \leq \alpha \cdot \min_{G \subseteq F:|G|=k} \max_{s \in [m]} c_p(\tilde{A}_s, \tilde{w}_{\tilde{A}_s}, G),
\]

then

\[
\max_{s \in [m]} c_p(A_s, w_{A_s}, \tilde{G}) \leq 3^p \cdot 2^{2p+2} \cdot \alpha \cdot \text{opt}.
\]

To prove this theorem, we need some definitions and results from other papers. We denote the cost of the \((\ell_p, k)\)-clustering on a set of clients \( A \subseteq \mathcal{A} \) with weights \( w : A \rightarrow \mathbb{R}_{\geq 0} \) and a set of centers \( F \subseteq \mathcal{F} \) with

\[
c_p(A, w, F) := \sum_{a \in A} w(a) \cdot d(a, F)^p.
\]

If the vector of weights is all one, we denote the cost simply by \( c_p(A, F) \).

**Definition 7.5.0.2 (Coreset).** For a weighted set of clients \( A \) with \( w_A : A \rightarrow \mathbb{R}_{\geq 0} \) and a set of centers \( F \) with size greater than \( k \), the set \( \tilde{A} \subseteq A \) with weights \( \tilde{w}_\tilde{A} : \tilde{A} \rightarrow \mathbb{R}_{\geq 0} \) is an \((\epsilon, k)\)-coreset for the \((\ell_p, k)\)-clustering objective if for any \( G \subseteq F \) of size \( k \),

\[
(1 - \epsilon)c_p(A, w_A, G) \leq c_p(\tilde{A}, \tilde{w}_\tilde{A}, G) \leq (1 + \epsilon)c_p(A, w_A, G)
\]

The above definition is also referred to as a strong coreset. A weak coreset is a coreset that gives the above guarantee only for optimal and near-optimal \( G \). The coreset definition is naturally extended to the socially fair clustering problem by replacing the cost function with \( \max_{s \in [m]} c_p(A_s, w_{A_s}, G) \). Note that in this case, we require a set \( \tilde{A}_s \subseteq A_s \), and weights \( \tilde{w}_{\tilde{A}_s} \) for each group \( s \in [m] \).

**Remark 7.5.0.3.** An \((\epsilon, k)\)-coreset with respect to a set of centers \( F \) is also an \((\epsilon, k)\)-
coreset with respect to \( \hat{F} \subseteq F \) with \( \hat{F} \geq k \).

**Theorem 7.5.0.4 ([295]).** There exists an algorithm that computes an \((\epsilon, k)\)-coreset of size \(O(k \log n/\epsilon^2 p)\) for the \((\ell_p, k)\)-clustering problem over a general metric.

The above result is achieved by the following sensitivity sampling approach that can be carried out in time \(O(nk \log n/\epsilon^2 p)\). Let \( F \) be a set of centers corresponding to a bicriteria approximation for the \((\ell_p, k)\)-clustering problem, i.e., \(|F| \geq k\). Then each point \( q \) is sampled with probability proportional to \( \frac{d(q, F)^p}{\sum_{v \in S_q} d(v, F)^p + \frac{1}{|S_q|}} \), where \( S_q \) is the set of points in the same cluster as \( q \) in the clustering corresponding to \( F \). The set of sampled points weighted by the inverse probability of selection gives the coreset. The following result shows how a coreset for the socially fair clustering problem can be found from an algorithm that finds a coreset for the \((\ell_p, k)\)-clustering problem.

**Theorem 7.5.0.5 ([294]).** Let \( \hat{A}_s \) with weights \( w_{\hat{A}_s} \) be an \((\epsilon, k)\)-coreset for the \((\ell_p, k)\)-clustering problem on a set of clients \( A_s \) with weights \( w_{A_s} \). Then \( \hat{A} := \hat{A}_1 \cup \cdots \cup \hat{A}_m \) is an \((\epsilon, k)\)-coreset for the socially fair \((\ell_p, k)\)-clustering problem on the set of clients \( A := A_1 \cup \cdots \cup A_m \).

The \( k \)-means++ starts by picking a facility uniformly at random and adding it to set \( F = \emptyset \). Then in the next steps it adds a point \( q \) to the set of facilities \( F \) with probability proportional to \( d(q, F) \). The following result states that the \( k \)-means++ algorithm finds a bicriteria approximation algorithm for the \((\ell_p, k)\)-clustering problem.

**Theorem 7.5.0.6 ([287]).** Let \( \beta > 1 \), \( F \) be a set of centers of size \( \lceil \beta k \rceil \) selected by the \( k \)-means++ algorithm, and \( \text{opt} \) be the optimal objective value for the \((\ell_p, k)\)-clustering problem. Then with high probability

\[
c_p(A, F) \leq \frac{2^{2p+1}}{1-\beta} \cdot \text{opt}.
\]
Corollary 7.5.0.7. Let \( \mathcal{A} = A_1 \cup \cdots \cup A_m \), \( \beta > 1 \), \( F_s \) be a set of centers of size \( \lceil \beta k \rceil \) selected by the \( k \)-means++ algorithm on the set \( A_s \). Moreover, let \( \text{opt} \) be the optimal objective value for the socially fair \((\ell_p, k)\)-clustering problem, and \( F = F_1 \cup \cdots \cup F_m \). Then

\[
\max_{s \in [m]} c_p(A_s, F) \leq \frac{2^{2p+1}}{1 - \beta} \cdot \text{opt}.
\]

Proof. Denoting the optimal objective value of the \((\ell_p, k)\)-clustering problem on \( A_s \) with \( \text{opt}_s \), by Theorem 7.5.0.6, we have

\[
c_p(A_s, F_s) \leq \frac{2^{2p+1}}{1 - \beta} \cdot \text{opt}_s \leq \frac{2^{2p+1}}{1 - \beta} \cdot \text{opt}.
\]

Moreover since \( F_i \subseteq F \), \( c_p(A_s, F) \leq c_p(A_s, F_s) \), and this implies the desired result. \( \square \)

We are now equipped to prove the main result of this section.

Proof of Theorem 7.5.0.1. For \( s \in [m] \), let \( \tilde{A}_s \) with weights \( w_{\tilde{A}_s} \) be an \((0.5, k)\)-coreset of size \( O(k \log n) \) for the \((\ell_p, k)\)-clustering problem on the set \( A_s \) with weights \( w_{A_s} \). Then by Theorem 7.5.0.5, \( \tilde{\mathcal{A}} := \tilde{A}_1 \cup \cdots \cup \tilde{A}_m \) is an \((0.5, k)\)-coreset for the socially fair \((\ell_p, k)\)-clustering problem on \( \mathcal{A} \). Therefore by definition, for all \( G \subseteq \mathcal{F} \) with \( |G| = k \),

\[
0.5 \max_{s \in [m]} c_p(A_s, w_{A_s}, G) \leq \max_{s \in [m]} c_p(\tilde{A}_s, w_{\tilde{A}_s}, G) \leq 1.5 \max_{s \in [m]} c_p(A_s, w_{A_s}, G)
\]

Now let \( F \) be a set of centers obtained by running the \( k \)-means++ algorithm with \( \beta = 2 \) on each \( A_s \) and taking the union. Then by Corollary 7.5.0.7, \( |F| = 2km \) and

\[
\max_{s \in [m]} c_p(A_s, w_{A_s}, F) \leq \frac{2^{2p+1}}{1 - \beta} \cdot \text{opt}.
\]

Let

\[
G^* = \arg \min_{G \subseteq \mathcal{F} : |G| = k} \max_{s \in [m]} c_p(A_s, w_{A_s}, G)
\]
Then by Lemma 7.2.0.5,
\[
\max_{s \in [m]} c_p(A_s, w_{A_s}, G^*) \leq 3^{p-1}(2^{2p+1} + 2) \cdot \text{opt}.
\]

Therefore by the coreset property
\[
\max_{s \in [m]} c_p(\hat{A}_s, w_{\hat{A}_s}, G^*) \leq 1.5 \cdot 3^{p-1}(2^{2p+1} + 2) \cdot \text{opt}.
\]

Finally by definition of \( \hat{G} \) and the coreset property,
\[
0.5 \cdot \max_{s \in [m]} c_p(A_s, w_{A_s}, \hat{G}) \leq \max_{s \in [m]} c_p(\hat{A}_s, w_{\hat{A}_s}, \hat{G}) \leq \alpha \max_{s \in [m]} c_p(\hat{A}_s, w_{\hat{A}_s}, G^*) \leq 1.5 \cdot 3^{p-1}(2^{2p+1} + 2) \cdot \alpha \cdot \text{opt}.
\]

\[\square\]

7.6 Empirical Study

We compare our algorithm with previously best algorithms in the literature on benchmark datasets for socially fair \( k \)-median problem. Namely, we compare our bicriteria algorithm with [74] (ABV), and our exact algorithm (that outputs exactly \( k \) centers) with [75] (MV). Since our bicriteria algorithm produces only a small number of extra centers (e.g., for two groups, our algorithm only produces one extra center — see Section 7.6.5), we search over the best \( k \)-subset in the set of \( k + m \) selected centers. However, instead of performing an exhaustive search combinatorially, we use a mixed-integer linear programming (MILP) solver to find the best \( k \)-subset. Although this is a very effective way to go from bicriteria approximation to a solution with exactly \( k \)-centers, our experiments show that running the MILP directly on the original problem is very slow. For example, on the Adult dataset with
10 groups and $k = 20$, the MILP solver did not converge after two hours.

Our code is written in MATLAB. We use IBM ILOG CPLEX 12.10 to solve the linear programs (and mixed-integer linear programs). For our experiments, we used a MacBook Pro (2019) with a 2.3 GHz 8-Core Intel Core i9 processor, a 16 GB 2667 MHz DDR4 memory card, a Intel UHD Graphics 630 1536 MB graphic card, 1 TB of SSD storage, and macOS version 12.3.1.

Datasets. We use three benchmark datasets that have been extensively used in the fairness literature. Similar to other works in fair clustering [224], we subsample the points in the datasets. Namely, we consider the first 500 examples in each dataset. A quick overview of the used datasets is in the following.

1. **Credit dataset** [223] consists of records of 30000 individuals with 21 features. We divided the multi-categorical education attribute to two demographic groups: “higher educated” and “lower educated.”

2. **Adult dataset** [268, 296] contains records of 48842 individuals collected from census data, with 103 features. We consider five racial groups of “Amer-Indian-Eskim”, “AsianPac-Islander”, “Black”, “White”, and “Other” for one of our experiments. For another experiment we consider the *intersectional* groups of race and gender (male and female) that results in 10 groups.

3. **COMPAS dataset** [254] is gathered by ProPublica and contains the recidivism rates for 9898 defendants. The data is divided to two racial groups of African-Americans (AA) and Caucasians (C).

Bicriteria approximation. The ABV algorithm, first solves the natural LP relaxation and then uses the “filtering” technique [297, 258] to round the fractional solution to an integral one. Given a parameter $0 < \epsilon < 1$, the algorithm outputs at most $k/(1 - \epsilon)$ centers and guarantees a $2/\epsilon$ approximation. In our comparison, we consider $\epsilon$ that gives almost the same number of centers as our algorithm. Tables in Section 7.6.5, summarise
Figure 7.2: Comparison of our bicriteria algorithm with ABV [74]. The number of centers our algorithm selects is close to $k$ and is often smaller than ABV (see Section 7.6.5).

the number of selected centers for different $k$ and $\epsilon$. The $\lambda$ parameter in our algorithm (see Algorithm 24 and Lemma 7.2.0.1) determines the factor of decrease in the radii of client balls in the iterative rounding algorithm.

As illustrated in Section 7.6.2, the performance of our algorithm do not change significantly by changing $\lambda$. So in our comparisons, we fix $\lambda = 0.6$. Fig. 7.2 illustrates that our algorithm outperforms ABV on different benchmark datasets. The gap between the performance of our algorithm and ABV becomes larger as the number of groups and $k$ become larger. For example, for the Adult dataset with 10 groups and $k = 50$, the objective value of ABV is almost twice of the objective that our algorithm achieves.

**Exactly $k$ centers.** The MV algorithm, first sparsifies the linear programming relaxation by setting the connection variables of points that are far from each other to zero. It then adopts a randomized rounding algorithm similar to [258] based on consolidating centers and points. In the process of rounding, it produces a $(1 - \gamma)$-restricted solution which is a solution where each center is either open by a fraction of at least $(1 - \gamma)$ or it is zero. The algorithm needs $\gamma < 0.5$. The results of MV for different values of $\gamma$ are presented in Section 7.6.2.

It appears that MV performs better for larger values of $\gamma$, so below we use $\gamma = 0.1$ and $\gamma = 0.4$ for our comparisons. Fig. 7.3 illustrates that our algorithms outperforms MV on different benchmark datasets. Similar to the bicriteria case, the gap between the performance of our algorithm and MV becomes larger as the number of groups and $k$
become larger. For example, for the Adult dataset with 5 or 10 groups and \( k = 50 \), the objective value of MV is almost thrice of the objective that our algorithm achieves.

### 7.6.1 Comparison of Algorithms Showing Both Maximum and Minimum

We compared different algorithms in terms of the objective value of the socially fair \((\ell_p, k)\)-clustering in the previous section. However, as we discussed previously, our underlying goal is to provide more equitable costs for different demographic groups. Therefore in this section, we illustrate the average cost for both the demographic group with maximum average cost and the demographic group with minimum average cost on solutions of different algorithms. The comparisons are illustrated in Figs. 7.4 and 7.5.

Figure 7.4: Comparison of our bicriteria algorithm with ABV [74]. The max and min on Subfigure (c) are across the demographic groups and are used to prevent cluttering plots with 5 groups. The number of centers our algorithm selects is close to \( k \) and is often smaller than ABV (see Section 7.6.5).
7.6.2 Results of Different Algorithms for Different Parameters

In this section, we investigate the dependency of the results of each algorithm to their corresponding parameter. These results are illustrated in Figs. 7.6 to 7.9. As suggested by results of these experiments, our algorithms are less sensitive to the choice of their corresponding parameter.

7.6.3 Comparison of Running Time of Different Algorithms in Practice

We compare the running time of our algorithms with that of ABV and MV on the three datasets (see Section 7.6.3). To summarize, the running times of our bicriteria algorithm and the exact algorithm with exhaustive search are virtually the same when the number of groups is no more than 5. Moreover our algorithms’ times are comparable to ABV and
Figure 7.7: Performance of our bicriteria algorithm of ABV [74] for different values of \( \epsilon \). The max and min on Subfigure (c) are across the demographic groups.

Figure 7.8: Performance of our algorithm with exactly \( k \) centers for different values of \( \lambda \). The max and min on Subfigure (c) are across the demographic groups.

Figure 7.9: Performance of the MV algorithm [75] for different values of \( \gamma \). The max and min on Subfigure (c) are across the demographic groups.

are significantly less than MV in most cases. The latter is because MV needs to guess the value of the optimal objective value. Therefore it needs to run the algorithm multiple times. We run the algorithm with 5 different values (by multiplying different factors of two) and output the best out of these for MV. These results are illustrated in Tables 7.1 to 7.4.
Table 7.1: Comparison of the running time of different algorithms on the first 200 samples of the Credit dataset averaged over five runs.

<table>
<thead>
<tr>
<th>k</th>
<th>Our Bicriteria ($\lambda = 0.2$)</th>
<th>Our Bicriteria ($\lambda = 0.6$)</th>
<th>ABV Bicriteria ($\epsilon = 0.3$)</th>
<th>ABV Bicriteria ($\epsilon = 0.4$)</th>
<th>Our Exact ($\lambda = 0.2$)</th>
<th>Our Exact ($\lambda = 0.6$)</th>
<th>MV Exact ($\gamma = 0.1$)</th>
<th>MV Exact ($\gamma = 0.2$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>11.12</td>
<td>13.20</td>
<td>12.21</td>
<td>11.07</td>
<td>11.56</td>
<td>13.28</td>
<td>33.42</td>
<td>30.41</td>
</tr>
<tr>
<td>25</td>
<td>13.20</td>
<td>13.67</td>
<td>11.51</td>
<td>12.20</td>
<td>14.06</td>
<td>14.51</td>
<td>34.04</td>
<td>36.52</td>
</tr>
<tr>
<td>30</td>
<td>4.42</td>
<td>10.32</td>
<td>4.59</td>
<td>3.64</td>
<td>4.53</td>
<td>10.40</td>
<td>20.59</td>
<td>15.28</td>
</tr>
<tr>
<td>35</td>
<td>3.60</td>
<td>3.74</td>
<td>4.53</td>
<td>3.89</td>
<td>4.27</td>
<td>5.88</td>
<td>21.97</td>
<td>23.35</td>
</tr>
<tr>
<td>40</td>
<td>5.57</td>
<td>3.86</td>
<td>4.23</td>
<td>3.62</td>
<td>7.92</td>
<td>6.09</td>
<td>16.33</td>
<td>19.77</td>
</tr>
</tbody>
</table>

Table 7.2: Comparison of the running time of different algorithms on the first 200 samples of the COMPAS dataset averaged over five runs.

<table>
<thead>
<tr>
<th>k</th>
<th>Our Bicriteria ($\lambda = 0.2$)</th>
<th>Our Bicriteria ($\lambda = 0.6$)</th>
<th>ABV Bicriteria ($\epsilon = 0.3$)</th>
<th>ABV Bicriteria ($\epsilon = 0.4$)</th>
<th>Our Exact ($\lambda = 0.2$)</th>
<th>Our Exact ($\lambda = 0.6$)</th>
<th>MV Exact ($\gamma = 0.1$)</th>
<th>MV Exact ($\gamma = 0.2$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>7.98</td>
<td>5.44</td>
<td>7.19</td>
<td>6.51</td>
<td>8.81</td>
<td>6.22</td>
<td>44.18</td>
<td>45.13</td>
</tr>
<tr>
<td>25</td>
<td>4.88</td>
<td>4.72</td>
<td>7.04</td>
<td>5.36</td>
<td>5.88</td>
<td>5.75</td>
<td>39.68</td>
<td>38.47</td>
</tr>
<tr>
<td>30</td>
<td>4.83</td>
<td>5.60</td>
<td>5.14</td>
<td>3.98</td>
<td>6.26</td>
<td>6.91</td>
<td>35.15</td>
<td>37.71</td>
</tr>
<tr>
<td>35</td>
<td>5.83</td>
<td>6.74</td>
<td>5.91</td>
<td>6.33</td>
<td>7.27</td>
<td>8.22</td>
<td>34.29</td>
<td>35.85</td>
</tr>
<tr>
<td>40</td>
<td>5.37</td>
<td>5.21</td>
<td>5.87</td>
<td>6.13</td>
<td>6.82</td>
<td>6.03</td>
<td>39.59</td>
<td>43.49</td>
</tr>
</tbody>
</table>

Table 7.3: Comparison of the running time of different algorithms on the first 200 samples of the Adult dataset with 5 race groups averaged over five runs.

<table>
<thead>
<tr>
<th>k</th>
<th>Our Bicriteria ($\lambda = 0.2$)</th>
<th>Our Bicriteria ($\lambda = 0.6$)</th>
<th>ABV Bicriteria ($\epsilon = 0.3$)</th>
<th>ABV Bicriteria ($\epsilon = 0.4$)</th>
<th>Our Exact ($\lambda = 0.2$)</th>
<th>Our Exact ($\lambda = 0.6$)</th>
<th>MV Exact ($\gamma = 0.1$)</th>
<th>MV Exact ($\gamma = 0.2$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>13.31</td>
<td>12.88</td>
<td>11.96</td>
<td>11.28</td>
<td>17.01</td>
<td>19.69</td>
<td>73.09</td>
<td>75.06</td>
</tr>
<tr>
<td>25</td>
<td>9.86</td>
<td>11.35</td>
<td>10.74</td>
<td>12.54</td>
<td>16.73</td>
<td>18.3</td>
<td>54.87</td>
<td>55.06</td>
</tr>
<tr>
<td>30</td>
<td>7.36</td>
<td>8.36</td>
<td>8.24</td>
<td>9.10</td>
<td>15.04</td>
<td>14.85</td>
<td>44.91</td>
<td>47.48</td>
</tr>
<tr>
<td>35</td>
<td>6.91</td>
<td>8.09</td>
<td>6.10</td>
<td>6.84</td>
<td>11.74</td>
<td>14.45</td>
<td>40.41</td>
<td>36.48</td>
</tr>
<tr>
<td>40</td>
<td>5.76</td>
<td>4.12</td>
<td>5.47</td>
<td>4.69</td>
<td>8.12</td>
<td>6.42</td>
<td>24.69</td>
<td>23.54</td>
</tr>
</tbody>
</table>

Table 7.4: Comparison of the running time of different algorithms on the first 500 samples of the Adult dataset with 10 race and gender groups averaged over five runs.

<table>
<thead>
<tr>
<th>k</th>
<th>Our Bicriteria ($\lambda = 0.2$)</th>
<th>Our Bicriteria ($\lambda = 0.6$)</th>
<th>ABV Bicriteria ($\epsilon = 0.3$)</th>
<th>ABV Bicriteria ($\epsilon = 0.4$)</th>
<th>Our Exact ($\lambda = 0.2$)</th>
<th>Our Exact ($\lambda = 0.6$)</th>
<th>MV Exact ($\gamma = 0.1$)</th>
<th>MV Exact ($\gamma = 0.2$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>166.2</td>
<td>159.7</td>
<td>136.5</td>
<td>125.5</td>
<td>414.3</td>
<td>212.5</td>
<td>597.0</td>
<td>686.2</td>
</tr>
<tr>
<td>25</td>
<td>171.7</td>
<td>195.0</td>
<td>165.3</td>
<td>149.2</td>
<td>250.3</td>
<td>254.4</td>
<td>753.3</td>
<td>679.2</td>
</tr>
<tr>
<td>30</td>
<td>185.1</td>
<td>150.3</td>
<td>152.6</td>
<td>163.5</td>
<td>335.3</td>
<td>153.5</td>
<td>690.1</td>
<td>690.5</td>
</tr>
<tr>
<td>35</td>
<td>176.5</td>
<td>176.3</td>
<td>178.4</td>
<td>175.9</td>
<td>265.4</td>
<td>183.3</td>
<td>786.8</td>
<td>795.5</td>
</tr>
<tr>
<td>40</td>
<td>168.7</td>
<td>170.2</td>
<td>157.6</td>
<td>181.4</td>
<td>223.6</td>
<td>199.7</td>
<td>867.8</td>
<td>816.6</td>
</tr>
</tbody>
</table>
7.6.4 Empirical Coreset Results on Full-Size Datasets

In this section, we present the results for running our algorithms, and algorithms of [74] and [75] on full-sized datasets using the coreset ideas discussed in Section 7.5. For each $k$, we consider three sets of randomly selected $k$-means++ centers and coresets. We run all algorithms on the same sets and we take the best one in terms of the objective function over the full datasets. The number of centers and the size of the coresets we pick are $1.2 \cdot k \cdot m$.

The running times are presented as the mean and standard deviation of the empirical running times over these three randomly selected coresets. The results demonstrate that our algorithms can be applied to full datasets using the ideas of Section 7.5, and they consistently outperform the algorithms of [74] and [75] both in terms of the achieved objective value and the running time. The results are illustrated in Figs. 7.10 and 7.11.

Figure 7.10: Results of the Credit dataset with 2 groups and 30,000 datapoints using the approach of Section 7.5.
7.6.5 Number of Selected Centers in The Bicriteria Algorithms

In this section, we compare the number of centers that our bicriteria algorithm selects compared to ABV. The results are illustrated in Tables 7.5 to 7.8. We have selected the appropriate parameters for comparison of the bicriteria algorithms using these results in order to compare parameters that lead to almost the same number of centers in practice.

7.7 Conclusion

We presented a polynomial time bicriteria algorithm for the socially fair ($\ell_p, k$)-clustering with $m$ groups that outputs at most $k + m$ centers. Using this, we presented two different constant-factor approximation algorithms that return exactly $k$ centers. An interesting future research is to investigate whether recent advancements in $k$-means++ [289, 290] could result in even faster constant-factor approximation algorithms for socially fair $k$-means.
Table 7.5: The number of selected centers for Abbasi-Bhaskara-Venkatasubramanian algorithm [74] (bottom table) and our bicriteria algorithm (top table) on the Credit dataset. \( \lambda \) is a parameter of our algorithm and denotes the amount of decrease in radii of balls around the clients in the iterative rounding algorithm. \( \epsilon \) is a parameter of ABV algorithm, and the maximum number of selected centers is \( k/(1 - \epsilon) \), which achieves a \( 2/\epsilon \) approximation factor.

<table>
<thead>
<tr>
<th>( k )</th>
<th>( \lambda = 0.2 )</th>
<th>( \lambda = 0.4 )</th>
<th>( \lambda = 0.6 )</th>
<th>( \lambda = 0.8 )</th>
<th>( \lambda = 1.0 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>10</td>
<td>11</td>
<td>11</td>
<td>11</td>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td>15</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>15</td>
<td>15</td>
</tr>
<tr>
<td>20</td>
<td>21</td>
<td>21</td>
<td>21</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>25</td>
<td>26</td>
<td>26</td>
<td>26</td>
<td>25</td>
<td>25</td>
</tr>
<tr>
<td>30</td>
<td>31</td>
<td>31</td>
<td>31</td>
<td>30</td>
<td>30</td>
</tr>
<tr>
<td>35</td>
<td>36</td>
<td>36</td>
<td>36</td>
<td>35</td>
<td>35</td>
</tr>
<tr>
<td>40</td>
<td>41</td>
<td>40</td>
<td>40</td>
<td>40</td>
<td>40</td>
</tr>
<tr>
<td>45</td>
<td>46</td>
<td>45</td>
<td>45</td>
<td>46</td>
<td>46</td>
</tr>
<tr>
<td>50</td>
<td>50</td>
<td>50</td>
<td>50</td>
<td>51</td>
<td>51</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( k )</th>
<th>( \epsilon = 0.1 )</th>
<th>( \epsilon = 0.2 )</th>
<th>( \epsilon = 0.3 )</th>
<th>( \epsilon = 0.4 )</th>
<th>( \epsilon = 0.5 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>10</td>
</tr>
<tr>
<td>10</td>
<td>11</td>
<td>12</td>
<td>14</td>
<td>16</td>
<td>20</td>
</tr>
<tr>
<td>15</td>
<td>16</td>
<td>18</td>
<td>21</td>
<td>23</td>
<td>29</td>
</tr>
<tr>
<td>20</td>
<td>20</td>
<td>23</td>
<td>26</td>
<td>29</td>
<td>34</td>
</tr>
<tr>
<td>25</td>
<td>22</td>
<td>27</td>
<td>31</td>
<td>35</td>
<td>39</td>
</tr>
<tr>
<td>30</td>
<td>29</td>
<td>34</td>
<td>37</td>
<td>41</td>
<td>46</td>
</tr>
<tr>
<td>35</td>
<td>25</td>
<td>29</td>
<td>36</td>
<td>40</td>
<td>45</td>
</tr>
<tr>
<td>40</td>
<td>26</td>
<td>30</td>
<td>38</td>
<td>43</td>
<td>48</td>
</tr>
<tr>
<td>45</td>
<td>36</td>
<td>40</td>
<td>46</td>
<td>53</td>
<td>59</td>
</tr>
<tr>
<td>50</td>
<td>38</td>
<td>45</td>
<td>51</td>
<td>58</td>
<td>64</td>
</tr>
</tbody>
</table>
Table 7.6: The number of selected centers for Abbasi-Bhaskara-Venkatasubramanian algorithm [74] (right table) and our bicriteria algorithm (left table) on the COMPAS dataset. $\lambda$ is a parameter of our algorithm and denotes the amount of decrease in radii of balls around the clients in the iterative rounding algorithm. $\epsilon$ is a parameter of ABV algorithm, and the maximum number of selected centers is $k/(1 - \epsilon)$, which achieves a $2/\epsilon$ approximation factor.

<table>
<thead>
<tr>
<th></th>
<th>$\lambda = 0.2$</th>
<th>$\lambda = 0.4$</th>
<th>$\lambda = 0.6$</th>
<th>$\lambda = 0.8$</th>
<th>$\lambda = 1.0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$k = 5$</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>$k = 10$</td>
<td>11</td>
<td>10</td>
<td>11</td>
<td>11</td>
<td>10</td>
</tr>
<tr>
<td>$k = 15$</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>$k = 20$</td>
<td>21</td>
<td>21</td>
<td>21</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>$k = 25$</td>
<td>26</td>
<td>26</td>
<td>25</td>
<td>26</td>
<td>26</td>
</tr>
<tr>
<td>$k = 30$</td>
<td>31</td>
<td>31</td>
<td>31</td>
<td>31</td>
<td>31</td>
</tr>
<tr>
<td>$k = 35$</td>
<td>36</td>
<td>36</td>
<td>36</td>
<td>36</td>
<td>36</td>
</tr>
<tr>
<td>$k = 40$</td>
<td>40</td>
<td>41</td>
<td>41</td>
<td>41</td>
<td>40</td>
</tr>
<tr>
<td>$k = 45$</td>
<td>46</td>
<td>46</td>
<td>46</td>
<td>46</td>
<td>46</td>
</tr>
<tr>
<td>$k = 50$</td>
<td>51</td>
<td>51</td>
<td>51</td>
<td>51</td>
<td>50</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>$\epsilon = 0.1$</th>
<th>$\epsilon = 0.2$</th>
<th>$\epsilon = 0.3$</th>
<th>$\epsilon = 0.4$</th>
<th>$\epsilon = 0.5$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$k = 5$</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>10</td>
</tr>
<tr>
<td>$k = 10$</td>
<td>11</td>
<td>12</td>
<td>14</td>
<td>15</td>
<td>16</td>
</tr>
<tr>
<td>$k = 15$</td>
<td>16</td>
<td>18</td>
<td>19</td>
<td>20</td>
<td>21</td>
</tr>
<tr>
<td>$k = 20$</td>
<td>22</td>
<td>24</td>
<td>24</td>
<td>25</td>
<td>27</td>
</tr>
<tr>
<td>$k = 25$</td>
<td>27</td>
<td>29</td>
<td>30</td>
<td>31</td>
<td>32</td>
</tr>
<tr>
<td>$k = 30$</td>
<td>30</td>
<td>34</td>
<td>35</td>
<td>36</td>
<td>38</td>
</tr>
<tr>
<td>$k = 35$</td>
<td>35</td>
<td>40</td>
<td>40</td>
<td>42</td>
<td>43</td>
</tr>
<tr>
<td>$k = 40$</td>
<td>42</td>
<td>46</td>
<td>48</td>
<td>49</td>
<td>51</td>
</tr>
<tr>
<td>$k = 45$</td>
<td>45</td>
<td>50</td>
<td>51</td>
<td>52</td>
<td>53</td>
</tr>
<tr>
<td>$k = 50$</td>
<td>50</td>
<td>56</td>
<td>58</td>
<td>59</td>
<td>61</td>
</tr>
</tbody>
</table>
Table 7.7: The number of selected centers for Abbasi-Bhaskara-Venkatasesubramanian algorithm [74] (right table) and our bicriteria algorithm (left table) on the Adult dataset with 5 groups. $\lambda$ is a parameter of our algorithm and denotes the amount of decrease in radii of balls around the clients in the iterative rounding algorithm. $\epsilon$ is a parameter of ABV algorithm, and the maximum number of selected centers is $k/(1 - \epsilon)$, which achieves a $2/\epsilon$ approximation factor.

<table>
<thead>
<tr>
<th></th>
<th>$\lambda = 0.2$</th>
<th>$\lambda = 0.4$</th>
<th>$\lambda = 0.6$</th>
<th>$\lambda = 0.8$</th>
<th>$\lambda = 1.0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$k = 5$</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>$k = 10$</td>
<td>15</td>
<td>14</td>
<td>12</td>
<td>14</td>
<td>12</td>
</tr>
<tr>
<td>$k = 15$</td>
<td>18</td>
<td>15</td>
<td>18</td>
<td>14</td>
<td>19</td>
</tr>
<tr>
<td>$k = 20$</td>
<td>24</td>
<td>20</td>
<td>22</td>
<td>18</td>
<td>20</td>
</tr>
<tr>
<td>$k = 25$</td>
<td>30</td>
<td>27</td>
<td>27</td>
<td>25</td>
<td>28</td>
</tr>
<tr>
<td>$k = 30$</td>
<td>34</td>
<td>34</td>
<td>33</td>
<td>30</td>
<td>33</td>
</tr>
<tr>
<td>$k = 35$</td>
<td>38</td>
<td>38</td>
<td>38</td>
<td>35</td>
<td>38</td>
</tr>
<tr>
<td>$k = 40$</td>
<td>43</td>
<td>43</td>
<td>43</td>
<td>40</td>
<td>44</td>
</tr>
<tr>
<td>$k = 45$</td>
<td>45</td>
<td>47</td>
<td>47</td>
<td>45</td>
<td>46</td>
</tr>
<tr>
<td>$k = 50$</td>
<td>54</td>
<td>54</td>
<td>54</td>
<td>54</td>
<td>50</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>$\epsilon = 0.1$</th>
<th>$\epsilon = 0.2$</th>
<th>$\epsilon = 0.3$</th>
<th>$\epsilon = 0.4$</th>
<th>$\epsilon = 0.5$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$k = 5$</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>$k = 10$</td>
<td>6</td>
<td>7</td>
<td>10</td>
<td>11</td>
<td>12</td>
</tr>
<tr>
<td>$k = 15$</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
</tr>
<tr>
<td>$k = 20$</td>
<td>13</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>17</td>
</tr>
<tr>
<td>$k = 25$</td>
<td>15</td>
<td>17</td>
<td>19</td>
<td>20</td>
<td>22</td>
</tr>
<tr>
<td>$k = 30$</td>
<td>18</td>
<td>19</td>
<td>21</td>
<td>23</td>
<td>28</td>
</tr>
<tr>
<td>$k = 35$</td>
<td>22</td>
<td>25</td>
<td>29</td>
<td>32</td>
<td>37</td>
</tr>
<tr>
<td>$k = 40$</td>
<td>30</td>
<td>37</td>
<td>39</td>
<td>45</td>
<td>50</td>
</tr>
<tr>
<td>$k = 45$</td>
<td>45</td>
<td>51</td>
<td>57</td>
<td>61</td>
<td>68</td>
</tr>
<tr>
<td>$k = 50$</td>
<td>45</td>
<td>56</td>
<td>61</td>
<td>68</td>
<td></td>
</tr>
</tbody>
</table>
Table 7.8: The number of selected centers for Abbasi-Bhaskara-Venkatasubramanian algorithm [74] (bottom table) and our bicriteria algorithm (top table) on the Adult dataset with 10 groups. \( \lambda \) is a parameter of our algorithm and denotes the amount of decrease in radii of balls around the clients in the iterative rounding algorithm. \( \epsilon \) is a parameter of ABV algorithm, and the maximum number of selected centers is \( k/(1 - \epsilon) \), which achieves a \( 2/\epsilon \) approximation factor.

<table>
<thead>
<tr>
<th>( k = 5 )</th>
<th>( \lambda = 0.2 )</th>
<th>( \lambda = 0.4 )</th>
<th>( \lambda = 0.6 )</th>
<th>( \lambda = 0.8 )</th>
<th>( \lambda = 1.0 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>9</td>
<td>9</td>
<td>8</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>18</td>
<td>19</td>
<td>17</td>
<td>18</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>24</td>
<td>22</td>
<td>16</td>
<td>22</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>27</td>
<td>29</td>
<td>20</td>
<td>23</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>35</td>
<td>34</td>
<td>25</td>
<td>26</td>
<td></td>
</tr>
<tr>
<td>38</td>
<td>38</td>
<td>36</td>
<td>31</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>44</td>
<td>45</td>
<td>43</td>
<td>36</td>
<td>35</td>
<td></td>
</tr>
<tr>
<td>48</td>
<td>47</td>
<td>44</td>
<td>48</td>
<td>49</td>
<td></td>
</tr>
<tr>
<td>51</td>
<td>52</td>
<td>51</td>
<td>50</td>
<td>51</td>
<td></td>
</tr>
<tr>
<td>57</td>
<td>59</td>
<td>55</td>
<td>57</td>
<td>53</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( k = 5 )</th>
<th>( \epsilon = 0.1 )</th>
<th>( \epsilon = 0.2 )</th>
<th>( \epsilon = 0.3 )</th>
<th>( \epsilon = 0.4 )</th>
<th>( \epsilon = 0.5 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>5</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>9</td>
<td>11</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>12</td>
<td>14</td>
<td>15</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>15</td>
<td>16</td>
<td>18</td>
<td>22</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>20</td>
<td>23</td>
<td>28</td>
<td>31</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>26</td>
<td>29</td>
<td>33</td>
<td>38</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>35</td>
<td>40</td>
<td>42</td>
<td>46</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>46</td>
<td>53</td>
<td>56</td>
<td>61</td>
<td></td>
</tr>
<tr>
<td>39</td>
<td>48</td>
<td>55</td>
<td>62</td>
<td>69</td>
<td></td>
</tr>
</tbody>
</table>
CHAPTER 8
OPEN PROBLEMS AND FUTURE RESEARCH

In this chapter, we discuss some interesting directions for future research and open problems regarding the three topics of the thesis. Although there are many interesting problems in each area, we focus on problems that are directly related to our results or are natural extensions and generalizations. Each section of this chapter covers one of the topics.

8.1 Stability and Bit Complexity of Efficient Algorithms for Optimization

Tall cases for LPs and p-norm regression. [76] has shown that for matrices $A \in \mathbb{R}^{n \times d}$ with $d \leq n$, an LP problem of the form $\min_{A^\top x = b, x \geq 0} c^\top x$ can be solved in $\tilde{O}(\sqrt{d})$ iterations instead of $\tilde{O}(\sqrt{n})$ iterations. This led to many exciting works, with the most recent one achieving an algorithm with $\tilde{O}((nd + d^2) \cdot \log(W/\epsilon))$ arithmetic operations [298, 78, 79], where $W$ is a parameter bounding the absolute value of all the numbers in the problem. Even more recently [299] showed the $p$-norm minimization problems of the form $\min \|Ax - b\|_p$, for $p \geq 2$, can be solved in $\tilde{O}_p(d^{(p-2)/(3p-2)})$ iterations instead of $\tilde{O}_p(n^{(p-2)/(3p-2)})$. However, they do not analyze the number of arithmetic operations for their algorithm. It is very interesting to settle the bit complexity and running time of these algorithms. We believe that our techniques and results would be helpful for these, but due to the more complex nature of the inverse maintenance in these problems, further tools are required as well.

Weighted linear regression in matrix multiplication time. The works on shifted numbers [18] have shown that a linear system can be solved in time $\tilde{O}(n^\omega \cdot \ell)$ instead of $\tilde{O}(n^\omega \cdot \log(\kappa/\epsilon))$. As we discussed in Section 2.1.1, this approach leads to an algorithm for solving LPs (Theorem 2.1.1.5) that is faster than the algorithm rising from the inverse
maintenance approach of [6, 7] (Theorem 2.1.1.4) in some settings. However, if we consider the worst case, the latter approach is faster. The main reason for this is that we have to solve linear systems of the form $A^T WAx = g$ in each iteration of IPM. The running time of the algorithm based on shifted numbers has a linear dependence on the bit complexity of the matrix, but the dependence on the bit complexity of vector $g$ is $\ell_g/n$, where $\ell_g$ is the bit complexity of $g$. However, note that the multiplication with $W$ is changing the bit complexity of the matrix and the only bound we have for entries of $W$ comes from the inner and outer radius of the LP. Then an important problem is whether linear systems of the form $A^T WAx = g$, where $W$ is a diagonal matrix, can be solved in time $\tilde{O}(n^\omega \cdot (\ell_A + \ell_W + \ell_g/n))$. An immediate consequence of such a result is an algorithm for solving LPs faster than the approach of [6, 7] in the worst case. Moreover this might lead to faster algorithms for exact LP solvers.

**Inverse maintenance with shifted numbers.** A drawback of the shifted numbers approach is that it does not work with current techniques for inverse maintenance since it does not produce the inverse as one single explicit matrix. Developing inverse maintenance techniques for solving dynamically changing linear systems using shifted number would improve the running times in Theorem 2.1.1.5 for solving LPs.

**Inverse maintenance with the sparse solver.** We showed that the $p$-norm minimization problem could be solved faster than matrix multiplication for sparse polyconditioned matrices for the current value of $\omega \approx 2.372$. However, for LPs, we can only show such a result for values of $\omega > 2.5$. This is mainly due to the bit complexity of the sparse solver that does not allow solving a large batch (of size close to $n$) of linear systems faster than matrix multiplication time and consequently prevents inverse maintenance for LPs for the current value of $\omega$. An approach to resolve this is to find a representation of the inverse in the sparse inverse solver with bit complexity $\tilde{O}(1)$. This then allows for solving a large batch of linear systems. We discussed one approach for achieving this in Chapter 3 based
on symmetric factorization of the inverse of Hankel matrices. However other approaches can be employed to achieve the same result. For example, if there exists an algorithm that multiplies two $n \times n$ Hankel matrices with bit complexity $\ell$ in $o(n^{\omega-1} \cdot \ell)$ time, then we can even compute an explicit inverse faster-than-matrix-multiplication time using the approach of Peng and Vempala [45].

**Bit complexity of general matrix data structures.** Very recently [300] has presented an approach for maintaining general matrix formulas. The general approach is that any matrix formula can be considered as a block of the inverse of some larger matrix. This is similar to the approach we utilized for maintaining $A(A^T X S^{-1} A)^{-1} A^T$ for linear programs. Exploring the bit complexity bounds in this general form would be interesting. The main questions here are the dependence of required error and condition number bounds on the input matrices and number of them.

### 8.2 Tensor Decompositions

**More efficient tensor completion.** In Chapter 4, we discussed the connection of Tucker decomposition to Kronecker product regression problems and used this to design very efficient algorithms for Tucker tensor decomposition. A related problem is tensor completion in which the error is only considered over a known subset of entries. This can be formulated using Tucker decomposition as the following. For a tensor $\mathcal{T} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$, and given a subset $S \subseteq [I_1] \times \cdots \times [I_N]$, the goal is to find $\mathcal{G} \in \mathbb{R}^{R_1 \times \cdots \times R_N}$, $A_1 \in \mathbb{R}^{I_1 \times R_1}$, $\ldots$, $A_N \in \mathbb{R}^{I_N \times R_N}$ that minimize

$$\sum_{(i_1, \ldots, i_N) \in S} [(\mathcal{T} - \mathcal{G} \times_1 A_1 \cdots \times_N A_N)_{i_1 \ldots i_N}]^2.$$

Then we can use the solution to complete the tensor, i.e., assign values to entries outside $S$. Then a related problem is the Kronecker product completion in which given a subset $S$ of
rows of $A_1 \otimes \cdots \otimes A_N$, the goal is to find $x \in \mathbb{R}^{R_1 \cdots R_N}$ that

$$
\text{minimize } \sum_{i \in S} [((A_1 \otimes \cdots \otimes A_N)x - b)_i]^2.
$$

In Chapter 4, we introduced an algorithm for the Kronecker product regression that is sublinear in the number of rows of $A_1 \otimes \cdots \otimes A_N$ and subquadratic in the number of columns of it. Our algorithm used the structure of leverage scores of $A_1 \otimes \cdots \otimes A_N$. However, for the completion problem, we lose such a structure on leverage scores. It is very interesting to explore the possibility of very efficient algorithms despite this.

**FPTAS for Tucker packing problem.** In Chapter 5, we introduced the Tucker packing problem that we used to find an approximately optimal shape for the core tensor of the Tucker decomposition. We showed Tucker packing problem is NP-hard and provided a PTAS for it. A more efficient approximation scheme is a fully polynomial-time approximation scheme (FPTAS) that asks for a dependence of $\text{poly}(\epsilon^{-1})$ for the error parameter $\epsilon$ of the problem (note that we want a $(1 - \epsilon)$-approximation). It is interesting to investigate whether the Tucker packing problem admits an FPTAS.

**More efficient algorithms for tree tensor packing problem.** We introduced the analogue of the Tucker packing problem for general tree tensor network decompositions in Section 5.5. We only provided a naive grid-search algorithm for approximately solving this problem. It is interesting to design more efficient approximation algorithms for this problem, e.g., PTAS. There are some challenges in designing more efficient algorithms for this problem. For example, even the number of singular values in the problem (size of the problem) might be exponential in $N$, the order of the tensor. A compelling question is whether we need to compute all of these singular values or we can only compute the larger ones and solve the problem using them.
8.3 Fair Clustering

Lloyd-like algorithms for socially fair $(\ell_p, k)$-clustering. In Chapter 6, we designed a very fast Lloyd-like algorithm for the geometric version of the socially fair $k$-means which corresponds to a structured $2$-norm regression problem. It is interesting to investigate whether one can design such algorithms for the geometric versions of socially fair $(\ell_p, k)$-clustering as well. This would be related to the $p$-norm regression problems we discussed in Chapter 2.

Existence of socially fair clustering. In Chapter 6, we presented the fair-Lloyd algorithm for socially fair $k$-means. A main component of fair-Lloyd is an algorithm that finds a fair set of centers given a $k$-clustering. The algorithm for this component can even check whether there exists a set of centers for the given $k$-clustering that achieves an equal average cost for the two demographic groups. It is interesting to see whether there exists a polynomial time algorithm that can check the existence of such a set of centers for the socially fair $k$-means, i.e., without checking all the possible clusterings of the data.

More efficient constant-factor approximation algorithms. In Chapter 7, we discussed algorithms with running time of $f(k, m) \text{poly}(n)$ for finding constant-factor approximate solution for socially fair $(\ell_p, k)$-clustering problems. An interesting question is the existence of constant-factor approximation algorithms that run in $f(m) \text{poly}(n, k)$ time.
REFERENCES


368


370


