

**MULTI-TREE MONTE CARLO METHODS FOR  
FAST, SCALABLE MACHINE LEARNING**

A Dissertation  
Presented to  
The Academic Faculty

by

Michael P. Holmes

In Partial Fulfillment  
of the Requirements for the Degree  
Doctor of Philosophy in  
Computer Science

College of Computing  
Georgia Institute of Technology  
May 2009

# MULTI-TREE MONTE CARLO METHODS FOR FAST, SCALABLE MACHINE LEARNING

Approved by:

Professor Charles Isbell, Advisor  
College of Computing  
*Georgia Institute of Technology*

Professor Alex Gray, Co-Advisor  
College of Computing  
*Georgia Institute of Technology*

Professor Frank Dellaert  
College of Computing  
*Georgia Institute of Technology*

Professor Rich Vuduc  
College of Computing  
*Georgia Institute of Technology*

Professor Michael Littman  
Department of Computer Science  
*Rutgers University*

Date Approved: December 18, 2008

# TABLE OF CONTENTS

LIST OF TABLES . . . . .	vi
LIST OF FIGURES . . . . .	vii
SUMMARY . . . . .	ix
I INTRODUCTION . . . . .	1
II FRAMING AND RELATED WORK . . . . .	6
2.1 Automatic Error Control . . . . .	6
2.2 Multi-Tree Methods . . . . .	8
2.2.1 Multi-Tree Summary . . . . .	12
2.3 Monte Carlo and Other Subsampling Methods . . . . .	13
2.3.1 Monte Carlo for Analytic Intractability . . . . .	14
2.3.2 Monte Carlo for Computational Speedup . . . . .	15
2.3.3 Monte Carlo Summary . . . . .	18
2.4 Combining Trees with Monte Carlo . . . . .	18
2.5 Note on Parallelization and Exact Algorithms . . . . .	20
III MONTE CARLO IN TREES: FAST KERNEL CONDITIONAL DENSITY ESTIMATION . . . . .	22
3.1 Introduction . . . . .	22
3.2 Kernel Conditional Density Estimation . . . . .	25
3.3 Bandwidth Selection . . . . .	27
3.3.1 Dual-Tree Approximation . . . . .	29
3.3.2 Dual-Tree for Fast Cross-Validated Likelihood . . . . .	31
3.3.3 Deterministic Approximation . . . . .	34
3.3.4 Monte Carlo Approximation . . . . .	37
3.3.5 Optimization . . . . .	39
3.4 Experiments . . . . .	39
3.4.1 Error Control and Speedup on Medium-Scale Datasets . . . . .	40

3.4.2	Scalability on Large Datasets . . . . .	42
3.5	Conclusion . . . . .	43
IV	TREES IN MONTE CARLO: FAST GENERAL KERNEL ESTIMATORS . . . . .	45
4.1	Introduction . . . . .	46
4.2	Problem Definition and Previous Work . . . . .	47
4.2.1	Previous Work . . . . .	49
4.3	Algorithmic Development . . . . .	51
4.3.1	Central Limit Conditions . . . . .	51
4.3.2	Base Case: Flat Monte Carlo . . . . .	53
4.3.3	General Case: Recursive Monte Carlo . . . . .	59
4.3.4	Full Algorithm: Recursive Monte Carlo with Tree-Stratified Sampling . . . . .	63
4.4	Experiments . . . . .	71
4.4.1	Error Control and Speedup on Medium-Scale Datasets . . . . .	72
4.4.2	Scalability on Large Datasets . . . . .	74
4.5	Conclusion . . . . .	76
V	RECIPROCAL TREES AND MONTE CARLO: FAST SINGULAR VALUE DECOMPOSITION . . . . .	84
5.1	Introduction . . . . .	85
5.2	Notation and SVD Background . . . . .	86
5.3	Previous Work . . . . .	90
5.3.1	Low-Rank Approximation with Additive Error Control . . . . .	90
5.3.2	Low-Rank Approximation with Relative Error Control . . . . .	93
5.3.3	Summary of LRMA Sampling Techniques . . . . .	95
5.4	QUIC-SVD: Whole-Matrix SVD Approximation with Relative Error Control . . . . .	97
5.4.1	Cosine Trees . . . . .	99
5.4.2	QUIC-SVD . . . . .	103
5.5	Empirical Results . . . . .	108

5.5.1	Cosine Trees vs. Previous Sampling Methods . . . . .	109
5.5.2	Error Control and Speedup on Medium-Scale Matrices . . .	114
5.5.3	Scalability on Large Matrices . . . . .	118
5.6	Conclusion . . . . .	121
VI	THE MULTI-TREE MONTE CARLO FRAMEWORK . . . . .	122
VII	CONCLUSION . . . . .	126
7.1	Future Extensions . . . . .	127
7.2	Implications for Machine Learning Applications . . . . .	129
7.3	Summary . . . . .	131
	REFERENCES . . . . .	133

## LIST OF TABLES

1	Results for deterministic dual-tree likelihood approximation over a range of medium-sized datasets. . . . .	41
2	Results for Monte Carlo dual-tree likelihood approximation over a range of medium-sized datasets. . . . .	41
3	Results for Monte Carlo dual-tree likelihood approximation on a set of large-scale datasets. . . . .	43
4	Descriptions of the six datasets used for medium-scale experiments, ranging from 17,605 to 50,000 in size and from 3 to 9 in dimension. . . . .	73
5	Distinctions between whole-matrix SVD approximation and low-rank matrix approximation. . . . .	98
6	Distinctions between subspace construction in QUIC-SVD and previous LRMA methods. . . . .	99
7	Descriptions of the six matrices used for medium-scale experiments, ranging from 3.6 million to 53 million entries in size. . . . .	110
8	Minimum and maximum relative squared error (as a fraction of $\epsilon$ ), across all runs at all error levels for each medium-scale matrix. . . . .	115
9	Descriptions of the three matrices used for large-scale experiments, ranging from 0.3 to 1.8 billion entries in size. . . . .	119

# LIST OF FIGURES

1	Structure of supporting evidence for thesis claims. . . . .	4
2	Estimating the distribution mean with a relative error tolerance of $\epsilon$ .	7
3	Dual-tree methods approximate the many pairwise relationships (e.g., distances) between points by a single pairwise relationship between the nodes containing the points. . . . .	9
4	Distribution $f(y, x)$ for which $f(y x)$ can be either bimodal or unimodal, depending on $x$ . The bold curve represents $f(y x = 80)$ . . . .	23
5	KCDE requires interpolation in both $x$ and $y$ . To estimate $\hat{f}(y x^*)$ , we place a kernel on each data point $(x_i, y_i)$ , and each of these kernels is weighted by a second kernel centered on $x^*$ . This gives a weighted set of kernels on the $y_i$ , which we use to construct the conditional estimate $\hat{f}(y x^*)$ . . . . .	27
6	In a dual-tree algorithm, we successively refine a tree-based partitioning of the data. At each point in the refinement, we test whether each pair of nodes can be approximated without further recursion. If the answer is yes, we use the approximation, otherwise we continue the refinement. . . . .	31
7	Mapping the kernel regression score $S_{KR}$ into the general case $G$ of the nested-summative formalism. . . . .	48
8	The RECURSIVEMC algorithm performs sampling at each level of nested summation. . . . .	59
9	Stratification can result in substantially lower per-stratum skewness, which yields low composite skewness. . . . .	66
10	Percentages of runs achieving their $\epsilon$ relative error target for RSMC and simple subsampling on $S_{KDE}$ (upper) and $S_{KR}$ (lower). The target percentage is $1 - \delta = 0.95$ , as represented by the dashed line. . . . .	74
11	Speedup vs. relative error tolerance for $S_{KR}$ on the galaxy16M (upper) and quasar2M (lower) datasets. . . . .	75
12	Speedup vs. relative error tolerance for $S_{KDE}$ on the sp500 dataset. . .	78
13	Speedup vs. relative error tolerance for $S_{KDE}$ on the cadata dataset. . .	78
14	Speedup vs. relative error tolerance for $S_{KDE}$ on the spectral dataset. . .	79
15	Speedup vs. relative error tolerance for $S_{KDE}$ on the sdss dataset. . . .	79
16	Speedup vs. relative error tolerance for $S_{KDE}$ on the quasar50K dataset. . .	80

17	Speedup vs. relative error tolerance for $S_{KDE}$ on the galaxy50K dataset.	80
18	Speedup vs. relative error tolerance for $S_{KR}$ on the sp500 dataset. . .	81
19	Speedup vs. relative error tolerance for $S_{KR}$ on the cadata dataset. . .	81
20	Speedup vs. relative error tolerance for $S_{KR}$ on the spectral dataset. . .	82
21	Speedup vs. relative error tolerance for $S_{KR}$ on the sdss dataset. . . .	82
22	Speedup vs. relative error tolerance for $S_{KR}$ on the quasar50K dataset.	83
23	Speedup vs. relative error tolerance for $S_{KR}$ on the galaxy50K dataset.	83
24	A cosine tree in two dimensions. . . . .	102
25	Schematic illustration of the QUIC-SVD algorithm. . . . .	105
26	$k_\epsilon$ is determined by the spectrum of the matrix, as represented here by a graph of the relative squared error of $AV_kV_k^T$ vs. $k$ . . . . .	108
27	Relative squared projection error vs. subspace rank for mars. . . . .	111
28	Relative squared projection error vs. subspace rank for madelon-kernel.	111
29	Relative squared projection error vs. subspace rank for declaration. . .	112
30	Relative squared projection error vs. subspace rank for mnist. . . . .	112
31	Relative squared projection error vs. subspace rank for arcene. . . . .	113
32	Relative squared projection error vs. subspace rank for galaxy. . . . .	113
33	Speedup vs. error tolerance for mars. . . . .	115
34	Speedup vs. error tolerance for madelon-kernel. . . . .	116
35	Speedup vs. error tolerance for declaration. . . . .	116
36	Speedup vs. error tolerance for mnist. . . . .	117
37	Speedup vs. error tolerance for arcene. . . . .	117
38	Speedup vs. error tolerance for galaxy. . . . .	118
39	Speedup vs. error tolerance for netflix-covar. . . . .	119
40	Speedup vs. error tolerance for zeta. . . . .	120
41	Speedup vs. error tolerance for dna-kernel. . . . .	120
42	Multi-Tree Monte Carlo meta-algorithm schematic. . . . .	123

## SUMMARY

As modern applications of machine learning and data mining are forced to deal with ever more massive quantities of data, practitioners quickly run into difficulty with the scalability of even the most basic and fundamental methods. We propose to provide scalability through a marriage between classical, empirical-style Monte Carlo approximation and deterministic multi-tree techniques. This union entails a critical compromise: losing determinism in order to gain speed. In the face of large-scale data, such a compromise is arguably often not only the right but the only choice. We refer to this new approximation methodology as Multi-Tree Monte Carlo. In particular, we have developed the following fast approximation methods:

1. Fast training for kernel conditional density estimation by injecting Monte Carlo into state-of-the-art dual-tree methods. Speedups as high as  $10^5$  have been shown on datasets of up to 1 million points.
2. Fast training for general kernel estimators (kernel density estimation, kernel regression, etc.) by injecting multiple trees into Monte Carlo. Speedups as high as  $10^6$  have been shown on tens of millions of points.
3. Fast singular value decomposition using a new form of sampling tree called the cosine tree. Speedups as high as  $10^5$  have been shown on dataset matrices containing billions of entries.

The level of acceleration shown by our methods represents improvement over the prior state of the art by several orders of magnitude. Such improvement not only speeds existing applications, it represents a qualitative shift, a commoditization, that

opens doors to all manner of new applications and method concepts that were previously invisible, outside the realm of practicality. Further, we show how the diverse operations of our approximation methods can be unified in a Multi-Tree Monte Carlo meta-algorithm which lends itself as scaffolding to the development of fast approximations for other methods we have not yet considered. Thus, our contribution includes not just the particular algorithms we have derived but also the Multi-Tree Monte Carlo methodological framework, which we hope will lead to many more fast algorithms that can provide the kind of scalability we have shown here to other important methods from machine learning and related fields.

# CHAPTER I

## INTRODUCTION

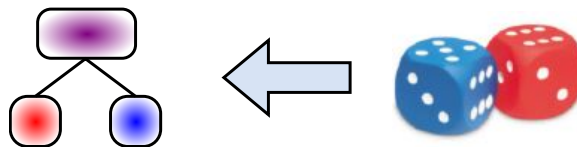
The goal of this work is to develop a methodology for constructing *scalable* machine learning algorithms that are capable of discovering *approximate solutions* with *user-controlled error*. The need for such algorithms is rapidly growing, as modern machine learning applications increasingly face massive amounts of data. Examples include: web search, commercial-scale collaborative filtering, large-scale network analysis (social, genetic, communications), processing the output of massive scientific experiments and simulations, activity discovery and recognition in large data streams (surveillance, finance, climatology), classification of medical imagery, and many more. The trend toward massive data is driven by the ever increasing proliferation of sensors, storage, and computerized interaction and inter-connectedness. The web includes over two billion pages which, at an average size of 1 KB, amounts to terabytes of data. The human genome contains around three billion base pairs. Wal-Mart's market-basket data is reported to be in the tens of terabytes, as is the Sloan Digital Sky Survey, and the Large Hadron Collider is expected to produce tens of terabytes daily [85, 78].

With such a profusion of information, it becomes essential to employ intelligent, automated methods to distill significant bits and insight. Machine learning, data mining, and related fields offer techniques that address this need. One controversial article went so far as to proclaim that such techniques, combined with loads of data, will lead eventually to “the end of theory” [3]. While that may overstate the case, it captures something of the trend. Machine learning and related methods are already recognized as a hot area, and will be in increasingly high demand for large-scale applications [96]. But we have a serious problem: machine learning doesn't scale.

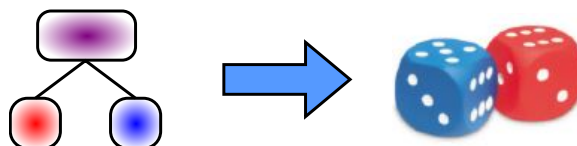
This is not to say that machine learning is intractable in the sense that its methods are not computable in polynomial time (though machine learning has its share of NP-hard problems [83, 11, 89, 23, 104, 24, 105]). But even low-order polynomial-time algorithms are impractical on massive datasets, where generally anything higher than  $O(n \log n)$  is too expensive. For instance, a  $O(n^3)$  algorithm on a dataset of size  $10^6$  would take (conservatively) over 10 years on a 3GHz processor. Runtimes of this order are quite common in machine learning [72], and few applications can tolerate methods with such poor scalability.

In this work, we derive scalable, approximate forms of several core methods used throughout machine learning:

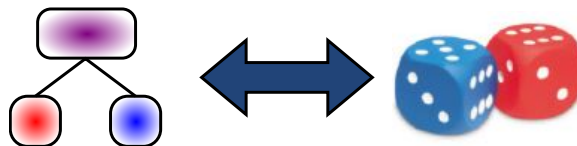
1. Fast **kernel conditional density estimator** training by putting Monte Carlo into state-of-the-art multi-tree methods.



2. Even faster training for **all kernel estimators** by putting data trees into Monte Carlo (includes kernel density estimation, kernel regression, kernel conditional density estimation, etc.).



3. Fast **singular value decomposition** by mutually-driven trees and Monte Carlo.



Our fast methods show speedups ranging from  $10^3$ – $10^6$  in large-data experiments, representing several-order-of-magnitude improvement over the prior state of the art. The methodology we introduce in these algorithms consists of a particular way of combining data partitioning trees with Monte Carlo-style sampling techniques. We refer to the methodology as *Multi-Tree Monte Carlo* (MTMC). MTMC grows primarily from the deterministic multi-tree methodology, in which speedup is obtained by exploiting geometric structure in the data [72, 106]. A key feature of multi-tree methods is *automatic error control*: the user sets a maximum allowed error, and the algorithm approximates within that tolerance. Contrast this with the more common approach in machine learning, in which computational effort is fixed (e.g., a set sample size) and error bounds are determined *post hoc*, if at all. Our approach retains the geometric and error-controlling advantages of the multi-tree methodology, but by combining trees with Monte Carlo techniques we are able to obtain higher speedups than prior methods, by several orders of magnitude. These high speedups are obtained at the cost of allowing the algorithms to become *controllably non-deterministic*.

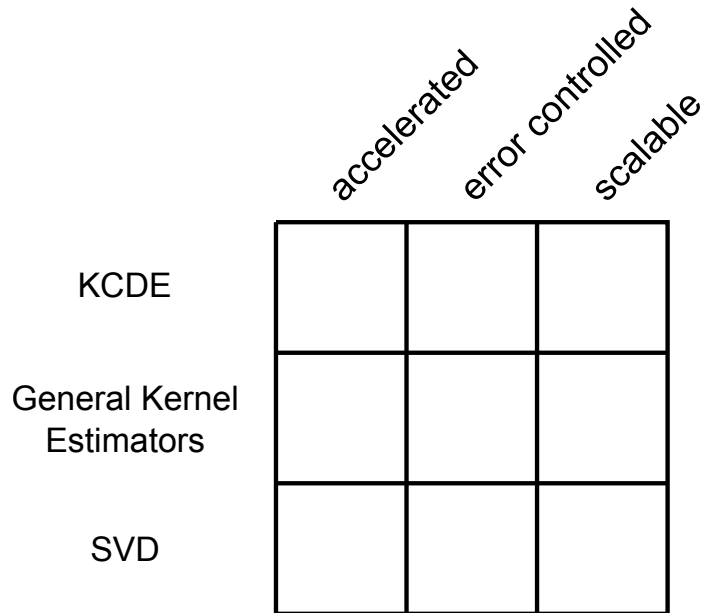
Based on these results, we argue for the following thesis:

**Thesis** — the *Multi-Tree Monte Carlo* methodology can be used to produce approximate machine learning methods that are:

1. Highly accelerated
2. Error-controlled
3. Scalable to large datasets

This thesis contains three subclaims, and we demonstrate each subclaim on each of our three methods as follows:

1. **Highly accelerated.** We demonstrate empirically that each of our methods achieves at least several-order-of-magnitude speedups across a broad array of dataset sizes and types.



**Figure 1:** Structure of supporting evidence for thesis claims.

2. **Error-controlled.** We derive theoretical error bounds and demonstrate empirically that they hold across a broad array of datasets.
3. **Scalable to large datasets.** We demonstrate efficient application of our methods on datasets with points and/or dimensions ranging from  $10^4$ – $10^6$ .

Structurally, our support for the thesis therefore consists of nine points of demonstration, as shown in Figure 1. We fill in the boxes by establishing each subclaim on each method, and this demonstrates the composite thesis.

Though this work provides high acceleration to an important set of machine learning methods, it is a mere opening of the door for the use of Multi-Tree Monte Carlo to produce fast and scalable algorithms. Other linear algebraic computations (e.g., solving linear systems, kernel matrix problems, non-negative matrix factorization, etc.), optimization problems (e.g., SVM optimization, semi-definite programming, etc.), and additional core methods are likely amenable to the same principles, and

this will be explored in future work. The present work, however, establishes the viability of Multi-Tree Monte Carlo for error-controlled speedup in machine learning. This methodology, in addition to the specific fast algorithms we provide, represents a significant contribution that could become the default approach to error-controlled algorithmic speedup in machine learning and related fields.

## CHAPTER II

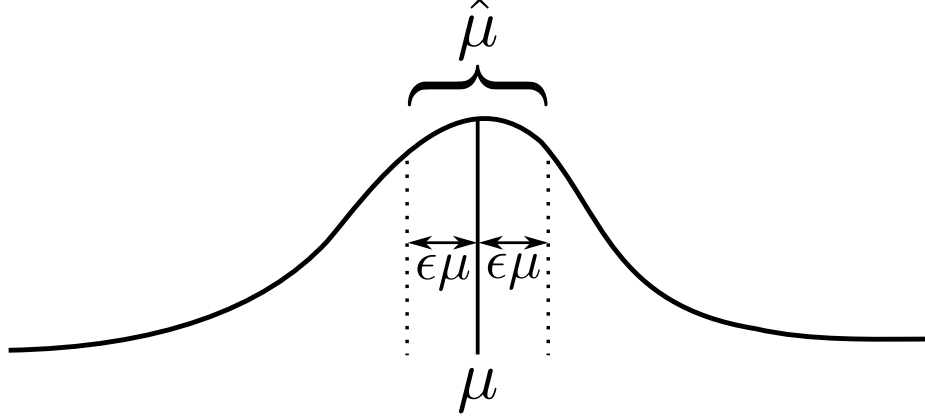
### FRAMING AND RELATED WORK

In this chapter we first lay out our approximation framework, which features the use of automatic, user-specified error control. We then describe the branches of previous work that are brought together in the Multi-Tree Monte Carlo methodology, which represents the convergence of deterministic multi-tree methods and subsampling/Monte Carlo techniques. Finally, we note the potential for parallelized versions of Multi-Tree Monte Carlo algorithms, while restricting the scope of this work to the sequential case.

#### *2.1 Automatic Error Control*

Our approach to approximation is distinguished from much of previous work by the type of user-specified error control we provide. Many approaches to approximate machine learning provide no error guarantees. Some provide asymptotic convergence guarantees with no characterization of finite-sample performance, and others put a cap on the computational effort (e.g., by taking a fixed sample size) while perhaps providing *post hoc* estimates of the induced error. Only very recently, with the advent of fast multi-tree methods, have approximate machine learning algorithms begun to provide strong user control of relative error. This is the framework we adopt.

More precisely, in this error control framework the user specifies an error tolerance  $\epsilon$ , and the approximation algorithm attempts to perform only the minimal work required to output an approximation within that tolerance. This is illustrated by Figure 2, in which an estimate  $\hat{\mu}$  of the mean  $\mu$  of a distribution is approximated to within the error bound  $|\mu - \hat{\mu}| \leq \epsilon|\mu|$ . Contrast this with the typical approach of approximating  $\mu$  by taking a sample set of fixed size  $k$ , and then perhaps estimating



**Figure 2:** Estimating the distribution mean with a relative error tolerance of  $\epsilon$ .

the error in the result. In our framework, the control knob is  $\epsilon$ , not  $k$ . This enables direct control of approximation quality.

While approximation in classical multi-tree methods also has this type of error control, our framework departs from that of the multi-tree work in that we allow the error bounds to hold probabilistically, rather than deterministically. For instance, in the case illustrated by Figure 2, an approximation algorithm in the MTMC framework might guarantee that  $|\mu - \hat{\mu}| \leq \epsilon|\mu|$  holds *with high probability* (e.g., with probability  $1 - \delta$ , with  $\delta$  controlled by the user). This relaxation allows the introduction of randomized algorithms that use more aggressive approximation to gain greater speed. Thus, our error control framework enables greater speedups than previous work, but at the cost of becoming controllably probabilistic in their output.

In sum, MTMC approximation algorithms provide error guarantees with the following properties:

1.  $\epsilon$  is set by the user.
2. Ideally, the error bound is *relative* to the magnitude of the object of estimation.
3. The bound need only hold probabilistically.

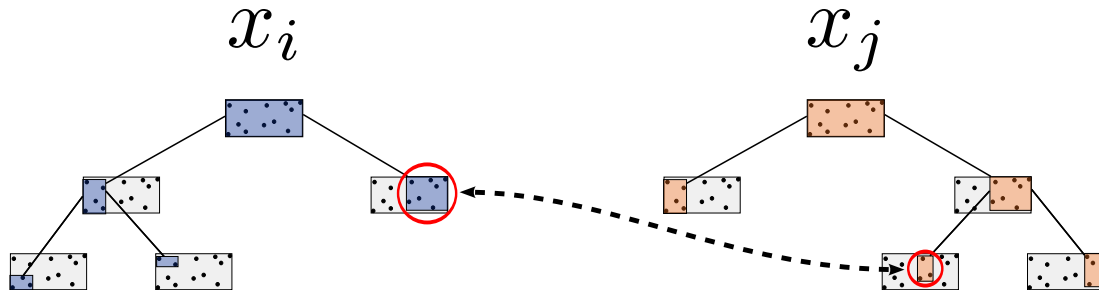
This automatic error control framework allow us to bring together two major veins

of work: 1) deterministic multi-tree methods, and 2) Monte Carlo and other sampling-based techniques. This takes various forms in our algorithms: putting Monte Carlo into trees, putting trees into Monte Carlo, or reciprocal interaction between trees and Monte Carlo. All of these modalities are subsumed in the general meta-algorithm of the Multi-Tree Monte Carlo methodology, as will be described in Chapter 6.

From the classical multi-tree approach, we inherit the basic idea of automatic relative error control and the use of spatial partitioning to help provide it. From the Monte Carlo approach, we gain tighter but probabilistic approximations, which enable much larger speedups while requiring the probabilistic relaxation of our error control. By combining these, we trade determinism for speed and form a powerful methodology for developing scalable methods that can handle massive datasets while retaining strong quality control.

## ***2.2 Multi-Tree Methods***

The multi-tree methodology encompasses the idea of using spatial partitioning to accelerate the computation of statistical  $n$ -body-type problems of a form similar to but more general than those encountered in computational physics [74]. The notion of “generalized  $n$ -body problems” was introduced Gray and Moore [72, 71], along with the principle of “higher-order divide and conquer” for constructing fast algorithms of both exact and approximate varieties. In their most basic form,  $n$ -body problems are those that require the computation of all pairwise distances between points in a dataset. This includes computations such as: all nearest neighbors, kernel regression or density estimates, range counts (average number of points with a specified distance of each point), and so on. In the case of pairwise distances, the multi-tree approach involves constructing dual spatial partitioning trees over the dataset, and using bounding box information from the trees to reason about all pairwise distances between pairs of nodes (one from each tree), rather than between all pairs of points.



**Figure 3:** Dual-tree methods approximate the many pairwise relationships (e.g., distances) between points by a single pairwise relationship between the nodes containing the points.

Figure 3 illustrates this approach. Speedup and error bounds come from reasoning about entire chunks (nodes) of data at a time. Leveraging such geometrical information can produce significant speedups, even for some exact computations.

More generally,  $n$ -body problems can involve arbitrary relationships between all  $k$ -tuples of points, rather than just all pairwise distances. The dual-tree idea then generalizes to  $k$  trees, and  $k$ -tree algorithms must reason about  $k$ -tuples of nodes at a time. Brute-force evaluation of a problem over all  $k$ -tuples of points requires  $O(n^k)$  computation, but multi-tree algorithms are conjectured to reduce this to  $O(n^{\log k})$ .

Algorithms in the multi-tree methodology either perform exact computations or return approximations with deterministically guaranteed error. Error guarantees can be either relative or absolute, and are user-controlled in the sense that the user specifies a desired error tolerance, and the algorithms try to perform the minimal work required to produce an approximation within that tolerance.

Since its introduction, the multi-tree methodology has been used to accelerate a variety of methods and applications. Much of the early work focused on fast kernel density estimation (KDE), a method important throughout machine learning, statistics, and many scientific applications. This work focused on both the optimization and evaluation phases of KDE, as described by Gray and Moore [66, 73, 67]. Speedups

on the order of  $10^2$ – $10^3$  were obtained on datasets ranging up to  $10^5$ – $10^6$  points. Empirically, realized error levels were reported far below their targeted  $\epsilon$  values. This highlights the fact that error bounding in deterministic multi-tree methods is accomplished by use of highly-conservative, worst-case bounds derived from the tree-node bounding surfaces. This is a general property: deterministic multi-tree algorithms are more conservative than is actually necessary to achieve targeted error levels.

The KDE work was expanded by Lee and Gray to include the use of multipole expansions in [98, 97]. Specifically, they use variants of the Fast Gauss Transform (FGT). The FGT is a type of series expansion for Gaussian kernels that yields tighter bounds within the dual-tree framework, taking out some of the overconservative slack and enabling faster approximation. Lee and Gray report speedups on the order of  $10$ – $10^2$  on 50K data points, including up to an order-of-magnitude improvement over the non-FGT dual-tree approach. One other empirical study is difficult to interpret, but seems to give similar results [95]. The FGT methods, however, only help within a limited regime: they do not scale with dimension and their advantage quickly falls off as dimension increases. In fact, all published multi-tree approaches to KDE acceleration have shown significantly decreasing speedup as dimension increases.

Raykar et al. propose a combination of the Improved Fast Gauss Transform with an adaptive space-partitioning scheme (a version of Gonzalez’ algorithm [63]) in a method they call IFGT [115, 117, 116]. Their IFGT allows for fast evaluation of weighted Gaussian sums, which they use to accelerate the computational primitive of kernel matrix-vector products. This is applied to several aspects of kernel density estimation, Gaussian process regression, and ranking problems. They sometimes combine IFGT with a  $k$ d-tree on cluster centers, which they refer to as FIGTree. Overall, they report speedups no higher than order  $10^3$ , and more frequently on the order of  $10$ – $10^2$ , using datasets on the order of  $10^3$ – $10^4$  points. Speedups are highest for data with very few (1-5) dimensions, and fall off very quickly with increasing dimension, though

they do report beating straight dual-tree algorithms in some regimes. In addition to scaling poorly with dimension, a major limitation to this approach is the restriction to Gaussian forms.

Another cluster of early work concentrated on accelerating  $n$ -point correlations for astronomical applications, as reported by Gray et al. [106, 69, 68].  $N$ -point correlations are a key bottleneck in many cosmological analyses, and the multi-tree approach has produced speedups on the order of  $10^3$  on datasets containing  $10^7$  objects.

A new type of nonparametric classifier, Kernel Discriminant Analysis (KDA), has been accelerated by multi-tree techniques [70, 118]. KDA learns class-conditional densities using kernel density estimates, then gives a Bayesian MAP prediction of the class of any query point. Dual-tree methods are particularly advantageous here, as densities only have to be evaluated to the point of deciding one is larger than the other. Speedups appear to be on the order of  $10^2$ – $10^3$ , enabling KDA to be applied to a quasar identification dataset of 40 million points from the Sloan Digital Sky Survey.

Klaas, De Freitas et al. applied the dual-tree idea to accelerate the “sum-kernel” and “max-kernel” bottlenecks encountered in belief propagation, the Viterbi algorithm, and especially in particle filtering methods [94, 93, 92]. They report speedups as high as several orders of magnitude in one case, as well as significant qualitative performance improvements as, for instance, the faster particle filtering methods are able to use more particles.

A small amount of work has attempted to apply the multi-tree methodology to linear algebra computations. De Freitas et al. combine  $n$ -body techniques with Krylov solvers (e.g., Lanczos iteration), emphasizing the stability of the resulting algorithms in light of the errors allowed by  $n$ -body methods [26]. Krylov methods are used for least-squares and eigenvalues problems, and can be used in Gaussian process regression, manifold learning and dimensionality reduction, etc. Speedups of at least 2–3 orders of magnitude are reported for specific applications.

Two other works have specifically addressed the matrix/vector product bottleneck to which Gaussian process regression (GPR) learning can be reduced [101]. Gray [65] gives a dual-tree approach to GPR that is shown to scale to millions of points, though speedups are not specifically reported. Shen et al. [132] give a similar dual-tree formulation, as well as a single-tree algorithm for GPR prediction. They report speedups of 3–82 on dataset sizes of 18K–40K.

Mean shift, a density-based clustering method, was accelerated with a dual-tree algorithm by Wang et al. in [139]. Speedups of 10–10<sup>2</sup> are demonstrated, and the method is shown to be more stable and accurate (due to error bounding) than previous attempts at accelerating mean shift. Applications to image segmentation are shown.

A dual-tree hierarchical clustering method was described by March and Gray in [102]. At the heart of the method is an accelerated form of Boruvka’s minimum spanning tree algorithm. Speedups of 5–11 on  $\sim 10^5$  points are reported.

Lastly, many of these multi-tree computations share a common algebraic form in terms of abstracted operators over datasets. This has been recognized and formalized by Boyer, Riegel and Gray [17]. They define a formal class of generalized  $n$ -body problems, and give a general algorithm that performs multi-tree acceleration for the entire problem class. THOR, the code framework for the generalized algorithm, also features automatic parallelization. In the preliminary work published thus far, they show speedups of 1–2 orders of magnitude on datasets with points in the millions.

### 2.2.1 Multi-Tree Summary

Taken together, the body of multi-tree work is broad and contains diverse applications. We can, however, make a few observations about the strengths and weaknesses of the methodology as a whole. Strengths include:

1. Applicability to a large class of “generalized  $n$ -body problems.”
2. Leveraging of geometric structure in the data to minimize computational work.

3. Hard, deterministic error guarantees.
4. Speedups typically between  $10$ – $10^3$ .

On the other hand, there are some significant drawbacks to the multi-tree approach.

Three of the most prominent are:

1. It is forced to rely on overconservative deterministic error bounds whose worst-case looseness prevents more aggressive approximation that would give greater speed.
2. Deriving a new multi-tree method to accelerate a given algorithm can be a convoluted and time-consuming task, though the THOR framework [17] may ameliorate this for at least a subset of  $n$ -body problems.
3. The standard multi-tree approach is conjectured to reduce  $O(n^k)$  computations at best to  $O(n^{\log k})$ , which still leaves an unscalable  $O(n^2)$  computation for  $k$  as small as 4.

Our Multi-Tree Monte Carlo methodology seeks to directly alleviate these problems by injecting the bound-tightness and speed of Monte Carlo. In doing so, we retain most of the strengths of multi-tree methods, but must give up one important feature: determinism. In the massive scale that we are targeting, speed is crucial, and unscalable methods cannot be practically applied. Thus, trading determinism for speed makes for an acceptable sacrifice.

### ***2.3 Monte Carlo and Other Subsampling Methods***

Traditionally, Monte Carlo techniques have employed randomized sampling to approximate expressions that are *analytically intractable*, i.e., that are difficult or impossible to solve in closed form. For instance, Monte Carlo has been heavily used for combinatorics, intractable integrals, and computing statistics on complex simulation models

[76, 123, 61]. In contrast to this classical usage, we employ Monte Carlo to accelerate problems that are *computationally impractical*. That is, we address computations where direct, closed-form evaluation is straightforward, such as  $\sum_i \sum_j K(x_i, x_j)$ , but where direct evaluation over a large dataset takes an impractically long time. Many classical Monte Carlo techniques can still be brought to bear in this scenario, enabling us to scalably approximate computations that are naively unscalable.

### 2.3.1 Monte Carlo for Analytic Intractability

Monte Carlo methods in machine learning have principally been of the classical sort, that is, they have been used to handle situations of analytic intractability [5]. A principle use is found in Bayesian methods, where estimators based on the posterior distribution frequently require the evaluation of analytically intractable integrals. For instance, Bayesian methods often require expectations of the form  $\int \theta p(\theta|D) d\theta$  for the posterior mean of a parameter  $\theta$ , given the data  $D$ . An unbiased estimated of this expectation is  $\frac{1}{n} \sum_i \theta_i$ , where the  $\theta_i$  are drawn from the distribution  $p(\theta|D)$ . Various Monte Carlo techniques have been developed or borrowed for these kinds of approximations in machine learning, including diverse forms of Markov Chain Monte Carlo (MCMC), importance sampling, Gibbs sampling, etc. [5, 4, 79, 120, 119, 109, 60, 87, 145]. Typically, these approximations operate without explicit error bounds or guarantees, often employing some kind of convergence criterion to terminate sampling.

Similar intractable integral expectations appear often in the E step of EM algorithms [9, 30, 58, 59, 103, 29]. Approximating these by a single sample is known as stochastic EM, or Monte Carlo EM if several samples are drawn [20, 21, 142]. Again, this is generally done without any explicit error bounding or guarantees.

Sequential Monte Carlo methods, including the popular particle filtering approach, make use of Monte Carlo to render tractable the integrals encountered in Bayesian filtering [86, 37, 137, 25]. As noted above, particle filtering can be further accelerated

with  $n$ -body-style multi-tree techniques.

In reinforcement learning (RL) and other Markov-model-based learning problems, one often encounters the need to compute expectations and other statistics on distributions over paths. Because these paths are often infinite in length, and the distributions are often unknown properties of the environment, the desired statistics are intractable and must be estimated by sampling [141, 136]. Additional complexity is introduced by the need to balance sampling driven by the need to learn (exploration) with exploitation of what has already been learned [91].

There are other examples of Monte Carlo for analytically intractable machine learning problems, but these are some of the most prominent and suffice for conveying the breadth and essential characteristics of the approach. In any case, this style of Monte Carlo is not the most relevant to our problem of deriving approximation algorithms for large datasets.

### 2.3.2 Monte Carlo for Computational Speedup

Somewhat less attention has been paid to the use of Monte Carlo for alleviating computational unscalability. The most common usage, which in some sense represents the state of the art for scalable machine learning, is naive subsampling or “mini-batching.” The idea here is to take a uniform, small random subset of the data and run the unscalable algorithm on that small subset. By shrinking the dataset to arbitrarily small sizes, algorithms can be made to run arbitrarily fast; however, there is an inevitable and often severe loss of quality relative to running on the full dataset. Induced error relative to running on the full dataset is often not quantified, though in some cases it can be. When it is quantified, it is generally either 1) a *post hoc* estimate, i.e., the user fixes the size of the subsample and then gets whatever error level comes out, or 2) an indirect convergence or bound in terms of training loss. Our approach, in contrast and as mentioned above, is to automatically control the

direct error of the quantity being estimated to a level specified by the user. A major advantage of Monte Carlo approximation is that error bounds are typically much tighter than the worst-case, conservative bounds used in multi-tree methods. The tradeoff is that they only hold with some probability, rather than deterministically.

Stochastic gradient descent is a popular example of a subsampling-based algorithm [13, 14, 131, 16, 15]. The idea is to take a gradient that, in exact form, requires evaluation of a costly sum over the dataset, and approximate it using a subsample from the dataset. This approximation is generally unbiased, and the hope is that as the gradient is repeatedly approximated during the course of an optimization, errors will cancel each other out and convergence to the optimum will still occur. It can be shown that, under some conditions, this convergence is asymptotically guaranteed. Gradient approximations are sometimes done using a single sample, and when done using several samples the sample set is often referred to as a “mini-batch.”

This idea of stochastic or mini-batch approximation has applicability to many learning methods other than gradient descent, including Newton-type methods, conjugate gradient, backpropagation, SMO, and more [130, 129, 131, 6, 146, 12]. With respect to *on-line* learning in particular (i.e., where the algorithm gets one pass through the dataset, seeing one point at a time), it has been noted that “while [full-dataset training] may have the stronger theoretical foundation, [on-line training] may yield better results and is more commonly used” [144].

Though not much written about, naive subsampling is also very commonly used in the face of massive datasets [56]. That is, rather than cycle through subsamples in an iterative process like gradient descent, one runs the full exact learning algorithm on a small random subset of the data. In practice, the sample size is often hand-selected; however, some methods exist for relating the subsample size to quality measures. The quality measure of interest has generally been something indirect such as training loss.

John and Langley introduced the idea of “dynamic sampling,” by which they

mean sampling that takes into account the characteristics of the learning algorithm in determining whether the sample is sufficient [88]. Leave-one-out cross-validation is used to estimate when accuracy stops increasing as the data subsample grows. This is a meta-algorithm that can be used with any learner. Such generality is a plus, but leave-one-out cross-validation is expensive, the stopping criterion is brittle, and no actual error guarantee is provided.

Toivonen presented a method for learning association rules in large databases by mining rules on a uniform subsample, then verifying them on the rest of the data [138]. He provides a probabilistic guarantee of capturing a target fraction of the rules that would have been found by mining the full data directly.

Quinlan’s work on the ID3 decision tree learning algorithm described a technique called “windowing,” in which the learner operates on a subset of the data, then increases the size of the subset until training error is deemed sufficiently low [114]. Fürnkranz showed efficiency and noise-handling gains in an extension [55]. No guarantee is provided on the quality of this method relative to training on the full dataset.

Domingos and Hulten developed the idea of bounding loss on a data subset relative to a hypothetical infinite dataset, then, at each step of a learning algorithm, only using the number of data points required to get an answer indistinguishable from what would be produced by learning on the infinite dataset [35, 36]. A similar idea was presented by Scheffer and Wrobel [127]. While interesting, this approach presents obvious difficulties: most datasets seem unlikely to contain enough points to make such perfect learning possible, and even if they did, the number of points required might be so large that little speedup is gained. Also, the necessary bounds are quite complex to derive, and probably do not exist for all learning methods.

The last approach we mention is the “progressive sampling” framework described by Provost, Jensen and Oates [113]. In some sense this framework envelopes much of the above-mentioned work, as it generalizes the notion of increasing sample size

until model quality ceases to improve. Their analysis concerns itself principally with the schedule of increasing samples sizes, and shows that schedules with geometrically increasing sample sizes have a type of asymptotic optimality. Detecting model/quality convergence is also addressed, but left as an open question.

Also worth noting is the use of repeated subsampling as a variance reduction mechanism, as in bagging [19], or as a means for estimating the standard error of an estimator, as in the bootstrap [44]. The aim of these techniques, however, is to gain statistical leverage from repeated subsampling, rather than to obtain scalability through a single subsample.

### **2.3.3 Monte Carlo Summary**

While Monte Carlo for scalability has clearly found some practical use in machine learning, existing approaches present several points upon which we can improve:

1. Error bounds that are *post hoc*.
2. Error bounds that are not in terms of the quantities being approximated (e.g., those that are indirect in terms of training error).
3. No use of the geometric structure of datasets: uniform sampling is the rule.

We can leverage the tight, probabilistic bounds and efficient small-sample performance made possible by Monte Carlo, while adding the geometric leverage and automatic error control framework of multi-tree-style methods. This forms the basis for our Multi-Tree Monte Carlo methodology.

## ***2.4 Combining Trees with Monte Carlo***

The Multi-Tree Monte Carlo methodology is a particular way of combining trees and Monte Carlo (or, more generally, spatial partitioning and sampling) in an iterative process that efficiently reduces direct error until a target error tolerance is achieved.

MTMC breaks from classical multi-tree methods in its introduction of efficient randomized subsampling techniques, at the cost of introducing probabilistic outcomes. It retains the focus on automatic, user-specified error control relative to the quantities that are actually being approximated, but allows error guarantees to be probabilistic, rather than deterministic. As we will show, the efficient combination of trees and Monte Carlo enables MTMC to produce speedups several orders of magnitude higher than the previous state of the art in error-controlled approximate machine learning, as represented by multi-tree methods.

We note that various prior works have used some combination of trees and sampling, but in a very different sense from the MTMC focus on fast approximation with automatic error control. For instance, Liu, Motoda, and Yu use a simple scheme to sample from a  $kd$ -tree inside the *Relief* feature-learning algorithm [99, 100]. No theoretical characterizations are given, but some experiments show (unsurprisingly) that the increased representativeness of the samples allows equal performance at lower sampling levels. Pechenizkiy, Puuronen, and Tsymbal use essentially the same sampling technique to subsample datasets prior to running other feature selection algorithms such as PCA [110]. Again, there is no theoretical analysis, only a small set of experiments measuring the accuracy of a classifier after feature selection using various sampling schemes.

Dellaert, Kwatra, and Oh introduced mixture trees for multi-resolution density modeling [28]. These trees present a number of advantages, including efficient conditional sampling. Tree construction employs a probabilistic sampling scheme for determining to which children the points in a parent node will be allocated, which in a sense is a combination of Monte Carlo and trees. Though effective for their intended purpose, mixture trees are less relevant to the approximation algorithms we derive.

Rudoy and Wolfe derived a sampling scheme for multi-modal distributions, where the distributions are modeled as multi-scale mixtures of Gaussians in a  $kd$ -tree [124].

They show faster empirical convergence of quantities derived from these distributions in comparison to Gibbs and other sampling methods.

An importance-sampling-based optimization scheme using *kd*-trees to partition the parameter space is described by Hamalainen et al. [75], building on work by Kajiyama and Painter and Sloan [90, 108]. The idea seems promising, but unfortunately their limited experiments present no baseline comparisons, and no theoretical characterization is given.

Lastly, some work in particle filtering and Monte Carlo localization (an application of particle filtering to robotics) has employed trees in various ways. The deterministic multi-tree work of de Freitas et al. has already been mentioned. Castro and McKenzie also describe an approach that hierarchically organize particles so they can be focused on areas of the state space that need higher accuracy, thus enabling the use of fewer particles and therefore less computation [47]. There is no theoretical analysis, nor any notion of error control.

So, in sum, the methodology we present, with its particular way of interleaving trees with Monte Carlo techniques, is quite different from anything in prior work. Building primarily on the foundation of the multi-tree methodology, we inject the fast probabilism of Monte Carlo techniques, thereby enabling higher speedups by several orders of magnitude, at the cost of introducing non-determinism.

## ***2.5 Note on Parallelization and Exact Algorithms***

While our focus in this work is on direct algorithmic acceleration, it should be clear throughout the description of our methods that they can be straightforwardly parallelized. Ultimately, greatest scalability will most likely come from systems employing both parallelization and algorithmic acceleration in combination. Our choice to focus on direct algorithmic acceleration merely reflects where we think we can make the most immediate impact.

On a final note, we point out that some approaches to scalability employ exact (non-approximate) methods. The SMO procedure for support vector machine optimization is a prime example of this [112, 111]. All else being equal, if an exact solution can be obtained efficiently we prefer it over an approximation. Often, however, we encounter computations that are fundamentally hard and therefore admit no efficient exact solution. In those cases, approximation is our only hope for scalability. Further, approximation generally allows for much more aggressive speedup. Because we are addressing the case of massive datasets, speed is a crucial performance criterion, and we therefore argue that approximation is often the right compromise.

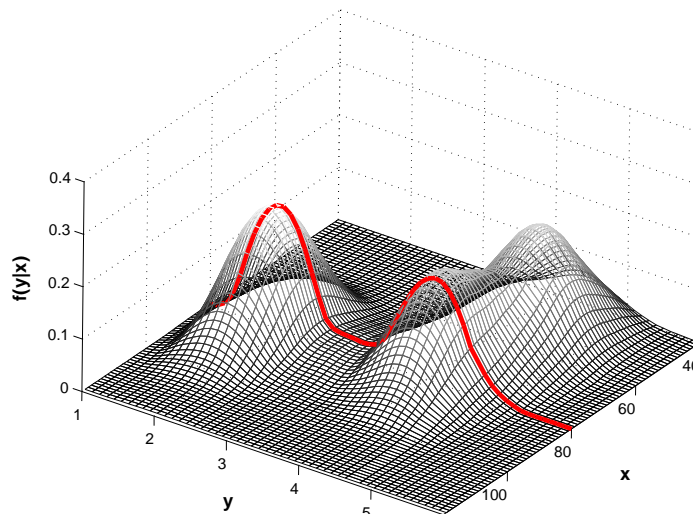
## CHAPTER III

### MONTE CARLO IN TREES: FAST KERNEL CONDITIONAL DENSITY ESTIMATION

Our first instantiation of the Multi-Tree Monte Carlo methodology centers on the idea of injecting Monte Carlo into state-of-the-art multi-tree-style methods. Specifically, we describe a fast learning method for kernel conditional density estimation (KCDE) [80]. While exact KCDE learning has an unscalable  $O(n^2)$  computational cost, our method is practical and shows speedups in the hundreds of thousands on datasets of one million points. First, we derive a classical, deterministic dual-tree approximation of the sort described in Chapter 2. We then replace the loose deterministic bounds with tight, probabilistic Monte Carlo bounds. The resulting Multi-Tree Monte Carlo algorithm exhibits strong error control and speedups of  $10^3$ – $10^5$  across a broad range of datasets. Using this fast algorithm, we are able to report the application of KCDE to large multivariate datasets many orders of magnitude greater in size than those reported in previous work. The cost of these large speedups is the loss of the strict error guarantee of the deterministic dual-tree framework; however, our experiments show that error is still well controlled by our Monte Carlo algorithm, and the many-order-of-magnitude speedups are worth this sacrifice in the large-data case, where learning for KCDE would otherwise be impossible.

#### ***3.1 Introduction***

Conditional density estimation is the estimation of the probability density  $f(y|x)$  of a random variable  $y$  given a random vector  $x$ . For example, in Figure 4 each contour line perpendicular to the  $x$  axis represents a conditional density. This can be viewed



**Figure 4:** Distribution  $f(y, x)$  for which  $f(y|x)$  can be either bimodal or unimodal, depending on  $x$ . The bold curve represents  $f(y|x = 80)$ .

as a generalization of regression: in regression we estimate the expectation  $E[y|x]$ , while in conditional density estimation we model the full distribution. Figure 4 illustrates a conditional bimodality such that  $E[y|x]$  is insufficiently descriptive for many tasks. Estimating conditional densities is much harder than regression, but having the full distribution is powerful because it allows one to extract almost any quantities of interest, including expectations, modes, prediction intervals, outlier boundaries, samples, expectations of non-linear functions of  $y$ , etc. Conditional densities also facilitate data visualization and exploration. Conditional density estimates are of fundamental and widespread utility throughout machine learning, and are applicable to such problems as time series prediction, continuous Markov models, Bayes nets and other graphical models, and static regression with prediction intervals. The estimation problem is challenging, however, because the data from which  $f(y|x)$  must be learned generally do not include any exact  $x$  for which  $f(y|x)$  will be queried.

Nonparametric kernel techniques address this issue by interpolating between the points that have been seen, without strong assumptions on the form of the distributions. In nonparametric conditional density estimation, we make only minimal

assumptions about the smoothness of  $f(y|x)$  without assuming any parametric form. Freedom from parametric assumptions is very often desirable when dealing with complex data, as we rarely have knowledge of true distributional structure. While a small amount of work on nonparametric kernel conditional density estimation has been done by statisticians and econometrics researchers [64, 49, 77, 8, 84, 121], it appears to have received little or no attention from the machine learning and computational statistics communities. Note that what we mean by nonparametric conditional density estimation is different from other techniques with similar names, such as conditional probability estimation (which refers to outputting class probabilities in the classification setting, also referred to as class-conditional probabilities) and various discrete and/or parametric conditional density models such as those commonly used in Bayes nets. The only machine learning work we have found that seems to look at the same problem is [126], but it employs a discretization scheme rather than handling continuous values directly.

In the present work, we use the standard kernel conditional density estimator that first received serious attention in the work of Fan et al. [48] and Hyndman et al. [84], though it was originally proposed by Rosenblatt [121]. This is a direct kernel estimator of conditional densities, as opposed to approaches that separately estimate  $f(y, x)$  and  $f(x)$ , which are combined to estimate  $f(y|x) = f(y, x)/f(x)$  [135]. Direct estimation of conditional densities allows the learning problem to be formulated as the optimization of a single, unified objective function, whereas separate estimation of  $f(y, x)$  and  $f(x)$  optimizes two different objective functions that may not align exactly with the objective of optimizing the quality of the *conditional* densities.

Although the direct estimator we use is consistent given mild conditions on its bandwidths, practical use has been hampered by the lack of an efficient data-driven bandwidth selection procedure, upon which any kernel estimator depends critically.

We propose a new method for efficiently selecting bandwidths to maximize cross-validated likelihood. The speedup of this method is obtained by combining Monte Carlo techniques with a dual-tree-based approximation [72] of the likelihood function. This approximation approach belongs to a new class of Multi-Tree Monte Carlo methods for scalable machine learning and statistics. Speeding up likelihood evaluations is also relevant for general nonparametric inference, but we focus here on the application to bandwidth selection. We present two versions of likelihood approximation, one analogous to previous dual-tree algorithms with deterministic error control, which gives speedups on the order of 1.5–10x in our experiments, and the other with a new, probabilistic Monte Carlo error control mechanism, which gives much larger speedups — as high as 286,000x on a million points.

With this fast learning procedure we are able to address datasets that are both higher in dimension and several orders of magnitude larger in size than those reported in any previous works, which were confined to bivariate datasets of size no greater than 1000 [49]. We present results that validate the accuracy and speedup of our likelihood approximation on datasets possessing a variety of sizes and dimensionalities. Most of these datasets were previously impractical to address with naively-computed data-driven techniques. Thus, our fast bandwidth optimization method enables applications at scales that were previously impossible. We conclude that kernel conditional density estimation is a powerful technique that is made substantially more efficient by our fast approximate optimization procedure, with many opportunities for application in machine learning and data analysis.

### ***3.2 Kernel Conditional Density Estimation***

We begin by drawing an analogy between the unconditional case of kernel density estimation (KDE), and the conditional case of KCDE. In KDE, we estimate a probability distribution  $f(x)$  from a dataset  $\{x_i\}$  by  $\hat{f}(x) = \frac{1}{n} \sum_i K_h(\|x - x_i\|)$ , where:

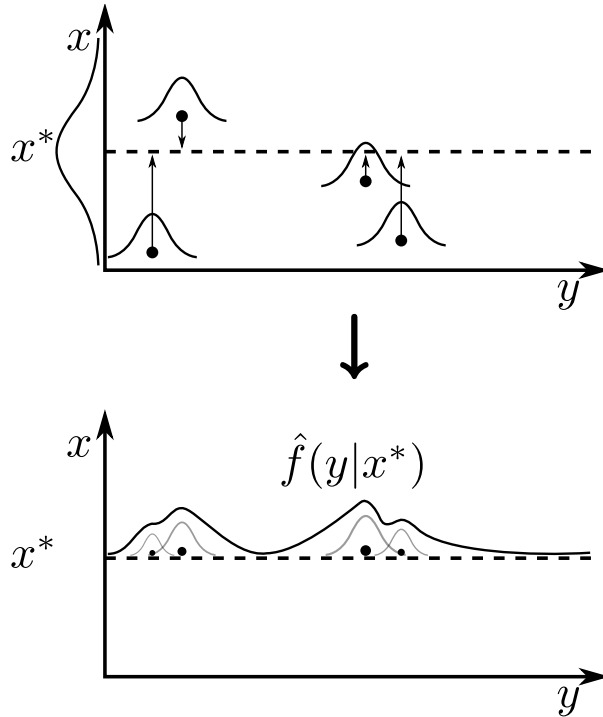
- $K_h(t) = \frac{1}{h^d} K(\frac{t}{h})$ ,
- $K$  is a kernel function, i.e., a compact, symmetric probability distribution such as the Gaussian or Epanechnikov kernels,
- $d$  is the dimension of  $x$ ,
- $n$  is the number of data points,
- $h$  is the bandwidth controlling the kernel widths (see [134]).

Kernels allow us to interpolate between the data we have seen in order to predict the density at points we haven't seen.

In kernel conditional density estimation, we estimate from a dataset  $\{x_i, y_i\}$  the set of all conditional distributions  $f(y|x)$ , rather than just  $f(x)$ . This means we have essentially a separate *KDE* problem for each value that  $x$  can take on. However, given a particular value  $x^*$  for which we wish to know  $f(y|x^*)$ , our dataset will most likely *not* contain any points with that precise value of  $x^*$ . Thus, for the individual KDE problem of that particular  $f(y|x^*)$ , we have no direct training data. A solution to this problem is to borrow the  $y_i$  training values that are paired with values  $x_i$  other than the queried  $x^*$ , and weight them according to how close their  $x_i$  are to  $x^*$ . This is illustrated in Figure 5. The borrowed  $y_i$  form a weighted training set on which something similar to KDE can be performed, finally yielding an estimate of  $f(y|x^*)$ . Because interpolation occurs in both  $x$  and  $y$ , this leads to a double kernel estimator:

$$\hat{f}(y|x) = \frac{\sum_i K_{h_1}(y - y_i) K_{h_2}(\|x - x_i\|)}{\sum_i K_{h_2}(\|x - x_i\|)}. \quad (1)$$

Because of its similarity to the Nadaraya-Watson kernel regression estimator, this form is known as the Nadaraya-Watson (NW) conditional density estimator [64]. For a queried  $x$ , it constructs a density by weighting each  $y_i$  proportionally to the proximity of the corresponding  $x_i$ . Note that there are now two bandwidths,  $h_1$  for the  $y$  kernel and  $h_2$  for the  $x$  kernel.



**Figure 5:** KCDE requires interpolation in both  $x$  and  $y$ . To estimate  $\hat{f}(y|x^*)$ , we place a kernel on each data point  $(x_i, y_i)$ , and each of these kernels is weighted by a second kernel centered on  $x^*$ . This gives a weighted set of kernels on the  $y_i$ , which we use to construct the conditional estimate  $\hat{f}(y|x^*)$ .

The NW estimator is consistent provided  $h_1 \rightarrow 0$ ,  $h_2 \rightarrow 0$ , and  $nh_1h_2 \rightarrow \infty$  as  $n \rightarrow \infty$  [84]. A few statisticians and econometrics researchers have made extensions to the NW estimator, most notably by the addition of local polynomial smoothing [48, 49, 64]. They have also proposed both reference rules and data-driven bandwidth selection procedures, but all applications appear to have been confined to the bivariate case, as has most of the theoretical analysis. One likely reason for this limitation is the difficulty of selecting good bandwidths in the presence of large datasets and higher dimensionality.

### 3.3 Bandwidth Selection

As with all kernel estimators, the performance of the NW estimator depends critically on a suitable choice for the bandwidths  $h_1$  and  $h_2$ . The aforementioned consistency

conditions provide little guidance in the finite-sample setting. Bandwidth selection has always been a dilemma: on the one hand, asymptotic arguments and reference distributions lead to plug-in and reference rules whereby bandwidths can be efficiently calculated, but these perform poorly on finite samples and when reference distributions don't match reality; on the other hand, data-driven selection criteria give good bandwidths but are naively intractable on datasets of appreciable size. We propose a middle road that captures some of the advantage of each approach by being both efficient and data-driven.

The only data-driven bandwidth score to previously appear in the KCDE literature is the integrated squared error in the following form:

$$ISE(h_1, h_2) = \int (f(y|x) - \hat{f}(y|x))^2 dy f(x) dx . \quad (2)$$

As shown in [49], minimizing ISE is equivalent to minimizing  $\int (\hat{f}(y|x))^2 dy f(x) dx - 2 \int \hat{f}(y|x) f(y, x) dy dx$ . A consistent, cross-validated estimate of the ISE is obtained by  $\widehat{ISE} = \frac{1}{n} \sum_i \int (\hat{f}^{-i}(y|x_i))^2 dy - \frac{2}{n} \sum_i \hat{f}^{-i}(y_i|x_i)$ , where  $\hat{f}^{-i}$  denotes  $\hat{f}$  evaluated with  $(x_i, y_i)$  left out.

Though appealing, the first term of  $\widehat{ISE}$  expands to a triply-nested summation, giving a base computational cost of  $O(n^3)$ . While this could still be used as the starting point for an efficient approximation, we choose to start with another criterion that has lower base complexity: likelihood cross-validation.

Likelihood cross-validation is well known in standard kernel density estimation (see [134], [73]), but has yet to be used for KCDE. One likely reason for this is the non-robustness to outliers that can afflict the likelihood function, particularly in the presence of heavy-tailed distributions. Although well known asymptotic results motivate the use of ISE instead of likelihood, we turn to the likelihood for this problem because of the significant computational benefit, balanced by our empirical observation that its performance is generally good.

By analogy with [134], we define the cross-validated log likelihood for KCDE to

be:

$$L(h_1, h_2) = \frac{1}{n} \sum_i \log(\hat{f}^{-i}(y_i|x_i)\hat{f}^{-i}(x_i)) , \quad (3)$$

where  $\hat{f}(x)$  is the standard kernel density estimate over  $x$  using the bandwidth  $h_2$  from  $\hat{f}(y|x)$ . We want to choose the bandwidth pair  $(h_1, h_2)$  that maximizes  $L$ ; by so doing, we will minimize the Kullback-Leibler divergence between our estimated density and the true density [134]. Furthermore, the likelihood score is naively computable in  $O(n^2)$  time, which gives us a better starting point for deriving a fast approximation algorithm.

### 3.3.1 Dual-Tree Approximation

Dual-tree recursion is a spatial-partitioning approach for error-controlled acceleration of computations that require the calculation of all pairwise distances. It has previously been used to accelerate a variety of computations such as finding all nearest neighbors, kernel density estimates, and two-point correlations. We present here a brief overview of the methodology, and refer the reader to the original papers for greater detail [72, 106, 73].

For a double summation  $\sum_i \sum_j g(x_i, x_j)$  over the data, the essential idea is that we can partition the set of pairs  $(x_i, x_j)$  into subsets within which the values  $g(x_i, x_j)$  are approximately constant. For each subset  $r$ , rather than explicitly compute  $g(x_i, x_j)$  for every pair, we can simply approximate it by  $\hat{g}_r$ . Speedup is gained by the fact that we do a single evaluation per subset, rather than evaluating all pairs; e.g., if there are  $n$  subsets, our computation becomes  $O(n)$  rather than  $O(n^2)$ . Error is controlled for the global computation by asking locally, for each subset  $r$ , whether the error induced by approximating all of the  $g(x_i, x_j)$  by  $\hat{g}_r$  is sufficiently low. If the answer is no, the subset is divided and the error test is repeated recursively.

---

**Algorithm 3.1** Generic dual-tree recursion.

---

DUALTREE

**Input:** nodes  $r_i$  and  $r_j$ ; error tolerance  $\epsilon$ **Output:** approximate contribution from  $r_i$  and  $r_j$  to overall computation

1. **if** CANAPPROXIMATE( $r_i, r_j, \epsilon$ )
    - (a) **return** APPROXIMATE( $r_i, r_j, \epsilon$ )
  2. **if** ISLEAF( $r_i$ ) and ISLEAF( $r_j$ )
    - (a) **return** DUALTREEBASE( $r_i, r_j$ )
  3. **else**
    - (a) **return** DUALTREE( $r_i.left, r_j.left, \epsilon$ )  $\oplus$  DUALTREE( $r_i.left, r_j.right, \epsilon$ )  
 $\oplus$  DUALTREE( $r_i.right, r_j.left, \epsilon$ )  $\oplus$  DUALTREE( $r_i.right, r_j.right, \epsilon$ )
- 

Suppose the data  $\{x_i\}$  are partitioned into subsets  $r \in R$ . We can write

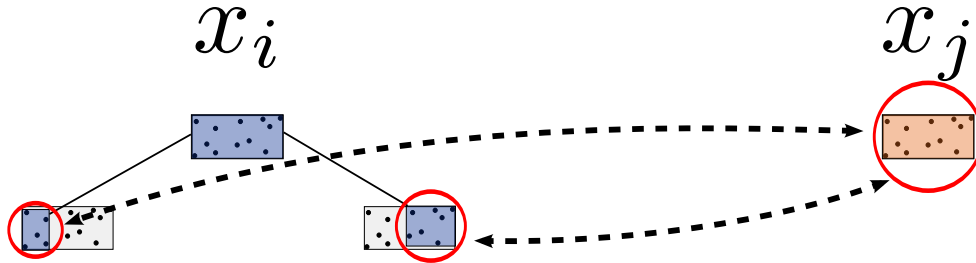
$$\sum_i \sum_j g(x_i, x_j) = \sum_{r_i \in R} \sum_{r_j \in R} g(r_i, r_j), \quad (4)$$

where  $g(r_i, r_j) = \sum_{x_i \in r_i} \sum_{x_j \in r_j} g(x_i, x_j)$ . If, for a given pair  $(r_i, r_j)$ , we can determine that  $g(x_i, x_j)$  lies within sufficiently narrow bounds for all  $x_i \in r_i$  and  $x_j \in r_j$ , then we can approximate by assuming all pairs in  $(r_i, r_j)$  have the same value, without calculating each term explicitly. The bounds on  $g(x_i, x_j)$  can be used to bound the error caused by this approximation.

In a dual-tree recursion (see Algorithm 3.1) we produce partitions  $R_i$  and  $R_j$  over  $\{x_i\}$  and  $\{x_j\}$  by traversing two separate *kd*-trees over the data.<sup>1</sup> See Figure 6 for an illustration.  $R_i$  and  $R_j$  contain the set of leaf nodes from each tree, and initially contain only the two root nodes. Note that every *kd*-tree node provides a tight bounding box for the points it contains. Starting with the roots, each call to the algorithm examines a node pair  $(r_i, r_j)$  to determine whether their contribution to the overall sum can be approximated (step 1). If so, we make the approximation and

---

<sup>1</sup>A *kd*-tree is a type of binary space partitioning tree used to speed up various kinds of computations. Other types of trees can also be used; see [106] for details.



**Figure 6:** In a dual-tree algorithm, we successively refine a tree-based partitioning of the data. At each point in the refinement, we test whether each pair of nodes can be approximated without further recursion. If the answer is yes, we use the approximation, otherwise we continue the refinement.

prune that branch of the recursion (step 1a). If not, the two nodes are each replaced by their children, resulting in four recursive node-node comparisons (step 3a), unless both nodes are leaves (step 2), in which case further refinement is impossible and we are forced to do the brute-force computation for that pair (step 2a). Note that the mechanism for merging the results of recursive subdivisions, as represented by the  $\oplus$  operator in the algorithm (step 3a), depends on the form of the function being approximated.

In order to apply the dual-tree methodology to a new problem, we must: 1) specify a node-node approximation function (APPROXIMATE in Algorithm 3.1), 2) specify the merge operator ( $\oplus$  in Algorithm 3.1), and 3) derive a “pruning rule” for deciding whether to approximate or recurse (CANAPPROXIMATE in Algorithm 3.1). In the dual-tree framework, approximation error should be guaranteed to fall within a threshold  $\epsilon$  set by the user. The approximation and pruning scheme must therefore be derived in such a way that local pruning decisions will lead to the desired global error guarantee.

### 3.3.2 Dual-Tree for Fast Cross-Validated Likelihood

We now derive a dual-tree-based approximation to the cross-validated likelihood  $L$ . Note that we use  $\Delta$  to denote the absolute error in the estimate of a quantity, e.g.,

$\Delta L \equiv |L - \widehat{L}|$ . The goal will be to guarantee the following error bound:

$$\Delta L \leq \epsilon , \quad (5)$$

where the error tolerance  $\epsilon$  is specified by the user. The main steps in deriving our algorithm are to specify the CANAPPROXIMATE and APPROXIMATE functions of Algorithm 3.1, along with an appropriate way of merging approximations from recursive subdivisions (step 3a of Algorithm 3.1). We begin by noting that, upon expansion of the  $\hat{f}^{-i}$  terms, we can write

$$\begin{aligned} L &= \frac{1}{n} \sum_i \log(\hat{f}^{-i}(y_i|x_i)\hat{f}^{-i}(x_i)) \\ &= \frac{1}{n} \sum_i \log \left[ \left( \frac{\sum_{j \neq i} K_{h_1}(y_i - y_j) K_{h_2}(\|x_i - x_j\|)}{\sum_{j \neq i} K_{h_2}(\|x_i - x_j\|)} \right) \left( \frac{1}{n-1} \sum_{j \neq i} K_{h_2}(\|x_i - x_j\|) \right) \right] \\ &= \frac{1}{n} \sum_i \log \left( \frac{\sum_{j \neq i} K_{h_1}(y_i - y_j) K_{h_2}(\|x_i - x_j\|)}{n-1} \right) \\ &= \frac{1}{n} \sum_i \log(A_i) - \log(n-1) , \end{aligned} \quad (6)$$

where  $A_i = \sum_{j \neq i} K_{h_1}(y_i - y_j) K_{h_2}(\|x_i - x_j\|)$ . The  $\log(n-1)$  term is constant and can be ignored. We will construct our approximation to  $L$  by first constructing a list of approximate  $A_i$  terms, whose logs we then sum to get an approximation  $\widehat{L}$ . In order to see how the various approximation errors propagate through to the total error in  $\widehat{L}$ , we will make use of the following exact error propagation bounds, which we state as a lemma.

**Lemma 1.** *Let  $f$  be a function  $f(x_1, \dots, x_n)$  of  $n$  arguments. If, for a given argument list  $(x_1, \dots, x_n)$  we substitute a list of approximate arguments  $(\hat{x}_1, \dots, \hat{x}_n)$  such that the absolute error  $|x_i - \hat{x}_i|$  in each  $x_i$  is bounded by  $\Delta x_i$ , the following bounds hold for  $\Delta f = |f(x_1, \dots, x_n) - f(\hat{x}_1, \dots, \hat{x}_n)|$ :*

1. *If  $f = \sum_i c_i x_i$ , then  $\Delta f \leq \sum_i c_i \Delta x_i$*
2. *If  $f = \log(x)$ , then  $\Delta f \leq \log(1 + \frac{\Delta x}{|x|})$ .*

We proceed by constructing a series of conditions, each of which implies  $\Delta L \leq \epsilon$ , and the last of which will be satisfied by our algorithm. Combining Rule 1 from Lemma 1 with Equation 6, we have  $\Delta L \leq \frac{1}{n} \sum_i \Delta \log(A_i)$ , so we can guarantee  $\Delta L \leq \epsilon$  by enforcing  $\frac{1}{n} \sum_i \Delta \log(A_i) \leq \epsilon$ , which is implied by  $\forall i, \Delta \log(A_i) \leq \epsilon$ . Invoking Rule 2 and a bit of rearrangement, this becomes equivalent to:

$$\forall i, \frac{\Delta A_i}{A_i} < e^\epsilon - 1. \quad (7)$$

Note that we drop the absolute value signs on  $A_i$  because, being a sum of kernel products, it is non-negative. Now consider a partitioning  $R$  of the data. We can write

$$A_i = \sum_{r \in R} \sum_{\substack{j \in r \\ j \neq i}} v(i, j) = \sum_{r \in R} S_{ir}, \quad (8)$$

where

$$v(i, j) \triangleq K_{h_1}(y_i - y_j) K_{h_2}(\|x_i - x_j\|) \quad (9)$$

$$S_{ir} \triangleq \sum_{\substack{j \in r \\ j \neq i}} v(i, j). \quad (10)$$

We therefore have  $\Delta A_i \leq \sum_r \Delta S_{ir}$ , and our enforcement condition holds if

$$\forall i, \frac{\sum_r \Delta S_{ir}}{\sum_r S_{ir}} \leq e^\epsilon - 1. \quad (11)$$

This condition is in turn implied by:

$$\forall i, r, \frac{\Delta S_{ir}}{S_{ir}} \leq e^\epsilon - 1. \quad (12)$$

Note that this is a stronger condition than necessary to imply Equation 11, but it is at least sufficient. We summarize this intermediate result as a lemma.

**Lemma 2.** *Let  $\widehat{L} = \frac{1}{n} \sum_i \log(\widehat{A}_i) - \log(n - 1)$  and let  $\widehat{A}_i = \sum_{r \in R} \widehat{S}_{ir}$  for some partitioning  $R$  of the data and some approximator  $\widehat{S}_{ir}$  of  $S_{ir}$ . If the approximations  $\widehat{S}_{ir}$  satisfy  $\forall i, r, \frac{\Delta S_{ir}}{S_{ir}} \leq e^\epsilon - 1$ , then  $\Delta L \leq \epsilon$ .*

### 3.3.3 Deterministic Approximation

Now consider two partitionings  $R_{outer}$  and  $R_{inner}$  induced by dual  $kd$ -trees on the dataset.  $R_{outer}$  represents a partitioning of the outer summation index  $i$ , while  $R_{inner}$  corresponds to the inner index  $j$  (see Equation 6). Recall that in the dual-tree framework (Algorithm 3.1), each partitioning comes from the leaves of a tree in its current state of expansion. At any point in the expansion, we compare a pair of nodes/partition elements  $r_i$  and  $r_j$ , each of which contains some subset of the indices. For each  $A_i$  with  $i \in r_i$ , the total contribution from all indices in  $r_j$  is  $\sum_{j \in r_j, j \neq i} v(i, j)$ . We can put bounds on the terms  $v(i, j)$  for all  $i \in r_i$  and  $j \in r_j$  using the bounding boxes of  $r_i$  and  $r_j$ . If the bounds on  $v(i, j)$  are tight enough that we can construct an approximation satisfying Lemma 2, then we can trigger CAN-APPROXIMATE for these contributions while staying consistent with the global error bound, otherwise we must invoke the recursive comparison of the children of  $r_i$  and  $r_j$ .

We now introduce the specific approximator  $\widehat{S}_{ir}$  we will use in our algorithm. Given a node pair  $r_i$  and  $r_j$  whose contribution is to be approximated, let  $v_{ij}^{min}$  and  $v_{ij}^{max}$  be bounds on  $v(i, j)$  (see Equation 9) given that  $i \in r_i$  and  $j \in r_j$ . Specifically, using distance bounds derived from the  $kd$ -tree bounding boxes, we have

$$v_{ij}^{min} = K_{h_1}(dy_{ij}^{max})K_{h_2}(dx_{ij}^{max}) \quad (13)$$

$$v_{ij}^{max} = K_{h_1}(dy_{ij}^{min})K_{h_2}(dx_{ij}^{min}) . \quad (14)$$

Define  $\hat{v}_{ij} = \frac{v_{ij}^{max} + v_{ij}^{min}}{2}$  to be the midpoint estimator for  $v(i, j)$  over  $r_i$  and  $r_j$ . Let  $n_{r_j}$  be the number of points in  $r_j$ . We estimate  $S_{ir_j}$  as follows:

$$\widehat{S}_{ir_j} = (n_{r_j} - 1)\hat{v}_{ij} . \quad (15)$$

This estimator corresponds to the APPROXIMATE function from Algorithm 3.1. We now give an intermediate lemma bounding the error of this approximation.

**Lemma 3.** Let  $\widehat{S}_{ir_j}$ ,  $v_{ij}^{max}$ , and  $v_{ij}^{min}$  be defined as in Equations 13–15. The absolute error  $\Delta S_{ir_j} = |S_{ir_j} - \widehat{S}_{ir_j}|$  satisfies  $\Delta S_{ir_j} \leq (n_{r_j} - 1) \frac{v_{ij}^{max} - v_{ij}^{min}}{2} + v_{ij}^{max}$ .

*Proof.* First note that setting  $\widehat{v}_{ij}$  to the midpoint of  $v_{ij}^{max}$  and  $v_{ij}^{min}$  means that no term  $v(i, j)$  in  $S_{ir_j}$  can differ from  $\widehat{v}_{ij}$  by more than  $\frac{v_{ij}^{max} - v_{ij}^{min}}{2}$ . With  $\widehat{S}_{ir_j} = (n_{r_j} - 1)\widehat{v}_{ij}$ , we have

$$\begin{aligned} \Delta S_{ir_j} &= |(n_{r_j} - 1)\widehat{v}_{ij} - \sum_{\substack{j \in r_j \\ j \neq i}} v(i, j)| \\ &\leq \max\left\{(n_r - 1) \frac{v_r^{max} - v_r^{min}}{2} + v_r^{max}, (n_r - 1) \frac{v_r^{max} - v_r^{min}}{2}\right\}. \end{aligned}$$

The first term in the max handles the case where  $i \notin r_j$ , in which case  $\widehat{S}_{ir_j}$  contains a  $\widehat{v}_{ij}$  for all but one of the  $v(i, j)$  terms in  $S_{ir_j}$ . We can therefore consider each matched term to be estimated with error no greater than  $\frac{v_{ij}^{max} - v_{ij}^{min}}{2}$ , and the unmatched term to be estimated as 0 with error no greater than  $v_r^{max}$ . The second term in the max handles  $i \in r$ , in which case all terms in  $S_{ir_j}$  are matched by terms in  $\widehat{S}_{ir_j}$ . The first term in the max is at least as large as the second, and the lemma follows.  $\square$

Lastly, we need to specify how to combine the answers from recursive subdivisions into a unified global approximation (i.e., the  $\oplus$  operator from Algorithm 3.1). Here we diverge somewhat from the typical dual-tree form in that we will keep our approximations  $\widehat{A}_i$  separate until the end, where the merge will happen by simply summing the  $\log \widehat{A}_i$  terms. Nonetheless, we use the dual-tree to share work across the  $\widehat{A}_i$ , in such a way as to guarantee the global error bound  $\Delta L \leq \epsilon$ . The resulting procedure DETLL is listed in Algorithm 3.2, and we now prove its correctness.

**Theorem 1.** Given a dataset  $D$ , the bandwidths  $h_1$  and  $h_2$ , and an error tolerance  $\epsilon$ , the DETLL algorithm returns an approximation  $\widehat{L}$  of the cross-validated log-likelihood  $L$  with respect to  $h_1$  and  $h_2$  over  $D$  such that the absolute error  $\Delta L$  satisfies  $\Delta L \leq \epsilon$ .

*Proof.* DETLL performs exact evaluation of the contributions to each  $A_i$  term in all cases except those where  $\frac{(n_{r_j} + 1)v_{ij}^{max}}{(n_{r_j} - 1)v_{ij}^{min}} \leq 2e^\epsilon - 1$ , in which case the contributions are

---

**Algorithm 3.2** Deterministic dual-tree approximation of the cross-validated log-likelihood objective.

---

DETL

**Input:** dataset  $D$  of points  $(x_i, y_i)$ ; bandwidths  $h_1$  and  $h_2$ ; error tolerance  $\epsilon$

**Output:**  $\hat{L}$  s.t.  $\Delta L \leq \epsilon$

1. Let  $\hat{A} = (0, 0, \dots, 0)$  contain initial estimates for all  $\hat{A}_i$
2. Let  $root_i = root_j = \text{CONSTRUCTKDROOT}(D)$
3.  $\text{DETDUALTREEAPPROX}(\hat{A}, root_i, root_j, \epsilon)$
4. Let  $\hat{L} = -\log(n - 1)$
5. **for**  $i = 0$  to  $n$ 
  - (a)  $\hat{L} += \frac{1}{n} \log \hat{A}_i$
6. **return**  $\hat{L}$

DETDUALTREEAPPROX

**Input:** list  $\hat{A}$  of partial  $\hat{A}_i$  values; nodes  $r_i$  and  $r_j$ ; error tolerance  $\epsilon$

**Output:** estimated contributions to  $\hat{A}_i$  from  $\forall i \in r_i, \forall j \in r_j$  (added to entries of  $\hat{A}$ )

1. **if**  $\frac{(n_{r_j}+1)v_{ij}^{max}}{(n_{r_j}-1)v_{ij}^{min}} \leq 2e^\epsilon - 1$ 
    - (a)  $\hat{S}_{ir_j} = (n_{r_j} - 1)\hat{v}_{ij}$
    - (b) **for**  $i \in r_i$ :  $\hat{A}_i += \hat{S}_{ir_j}$  // approximation of  $S_{ir_j}$
  2. **else if**  $\text{NOTSPLITTABLE}(r_i)$  and  $\text{NOTSPLITTABLE}(r_j)$ 
    - (a) **for**  $i \in r_i$ :  $\hat{A}_i += S_{ir_j}$  // exact evaluation of  $S_{ir_j}$
  3. **else**
    - (a) **for**  $(p, q) \in \{\{r_i.left, r_i.right\} \times \{r_j.left, r_j.right\}\}$ 
      - i.  $\text{DETDUALTREEAPPROX}(\hat{A}, p, q, \epsilon)$  // all pairs of child nodes
- 

approximated as  $\hat{S}_{ir_j} = (n_{r_j} - 1)\hat{v}_{ij}$ . The key to the proof is therefore to show that such approximation leads to  $\Delta L \leq \epsilon$ .

By Lemma 2, if all approximations  $\hat{S}_{ir_j}$  satisfy  $\frac{\Delta S_{ir_j}}{S_{ir_j}} \leq e^\epsilon - 1$ , then  $\Delta L \leq \epsilon$ . Since  $S_{ir_j} \geq (n_{r_j} - 1)v_{ij}^{min}$ , we have  $\frac{\Delta S_{ir_j}}{S_{ir_j}} \leq \frac{\Delta S_{ir_j}}{(n_{r_j}-1)v_{ij}^{min}}$ , so we can enforce the condition

of Lemma 2 by ensuring that  $\frac{\Delta S_{ir_j}}{(n_{r_j}-1)v_{ij}^{min}} \leq e^\epsilon - 1$ . By Lemma 3, approximating with  $\hat{S}_{ir_j} = (n_{r_j} - 1)\hat{v}_{ij}$  gives  $\Delta S_{ir_j} \leq (n_{r_j} - 1)\frac{v_{ij}^{max}-v_{ij}^{min}}{2} + v_{ij}^{max}$ . Substituting this bound and rearranging shows that the Lemma 2 condition is implied by enforcing  $\frac{(n_{r_j}+1)v_{ij}^{max}}{(n_{r_j}-1)v_{ij}^{min}} \leq 2e^\epsilon - 1$ . This exactly matches the way and conditions in which DETLL performs approximation. Thus, the approximation scheme of DETLL satisfies the conditions of Lemma 2, and the algorithm therefore guarantees  $\Delta L \leq \epsilon$ .  $\square$

### 3.3.4 Monte Carlo Approximation

While the deterministic pruning rule of DETLL guarantees the desired error bound, in practice it operates with a high degree of overconservatism due to the looseness of the deterministic bounds used in its derivation. In particular, some of the overconservatism stems from the fact that  $v_{ij}^{min}$  and  $v_{ij}^{max}$  are extreme values based on the corners of  $kd$ -tree bounding boxes. These bounds may be very far from the majority of the  $v(i, j)$  terms being summed. We therefore introduce a new Monte Carlo scheme that uses samples to get a better idea of the composition of the  $v(i, j)$  values contained in a given node pair. This scheme allows us to focus on the *typical* rather than the *extreme* case; it is also more robust to the presence of outliers. As a result, we end up with tighter bounds, higher prunes, and therefore larger speedups; however, the increase in speed comes at the cost of losing the strict, deterministic error guarantees of DETLL. As we will see, the overconservatism of the rest of the error bounding framework is such that, in practice, the added error from the Monte Carlo scheme is more than compensated for.

In particular, for the Monte Carlo scheme we go back to Lemma 2, which guarantees the  $\Delta L \leq \epsilon$  error bound provided that  $\frac{\Delta S_{ir_j}}{S_{ir_j}} \leq e^\epsilon - 1$ . Rather than estimate  $S_{ir_j}$  and  $\frac{\Delta S_{ir_j}}{S_{ir_j}}$  in terms of  $v_{ij}^{min}$  and  $v_{ij}^{max}$ , we instead collect a set of  $(i, j)$  pairs sampled from  $r_i$  and  $r_j$ , which we then use to construct a confidence interval for  $v(i, j)$  over the node pair. From this confidence interval we extract a tighter approximation for

---

**Algorithm 3.3** Monte Carlo dual-tree approximation of the cross-validated log-likelihood objective.

---

MCLL

**Input:** dataset  $D$  of points  $(x_i, y_i)$ ; bandwidths  $h_1$  and  $h_2$ ; error tolerance  $\epsilon$ ; sample size  $m$ ; number of sample replications  $B$ ; number of standard errors  $z$

**Output:**  $\hat{L}$ , targeting approximation error  $\Delta L \leq \epsilon$

identical to DETLL (Algorithm 3.2), with the exception of line 3:

3. MCDUALTREEAPPROX( $\hat{A}$ ,  $root_i$ ,  $root_j$ ,  $\epsilon$ ,  $m$ ,  $B$ ,  $z$ )

MCDUALTREEAPPROX

**Input:** list  $\hat{A}$  of partial  $\hat{A}_i$  values; nodes  $r_i$  and  $r_j$ ; error tolerance  $\epsilon$ ; sample size  $m$ ; number of sample replications  $B$ ; number of standard errors  $z$

**Output:** estimated contributions to  $\hat{A}_i$  from  $\forall i \in r_i$ ,  $\forall j \in r_j$  (added to entries of  $\hat{A}$ )

1. Sample  $m$  pairs  $(i, j)$  uniformly from  $r_i$  and  $r_j$ , rejecting if  $i = j$
2.  $\mu_v =$  mean of  $v(i, j)$  over sample,  $f_{skip} =$  fraction of pairs rejected due to  $i = j$
3.  $\sigma_\mu =$  standard error of  $\mu_v$  ( $\sigma/\sqrt{B}$  from  $B$  bootstrap resamplings of size  $m$ )
4. **if**  $z\sigma_v/\mu_v \leq e^\epsilon - 1$ 
  - (a)  $\hat{S}_{ir_j} = (1 - f_{skip})n_{r_j}\mu_v$
  - (b) **for**  $i \in r_i$ :  $\hat{A}_i += \hat{S}_{ir_j}$  // approximation of  $S_{ir_j}$
5. **else if** NOTSPLITTABLE( $r_i$ ) and NOTSPLITTABLE( $r_j$ )
  - (a) **for**  $i \in r_i$ :  $\hat{A}_i += S_{ir_j}$  // exact evaluation of  $S_{ir_j}$
6. **else**
  - (a) **for**  $(p, q) \in (\{r_i.left, r_i.right\} \times \{r_j.left, r_j.right\})$ 
    - i. MCDUALTREEAPPROX( $\hat{A}$ ,  $p$ ,  $q$ ,  $\epsilon$ ) // all pairs of child nodes

---

$\frac{\Delta S_{ir_j}}{S_{ir_j}}$ , which we then plug into the condition of Lemma 2 for the pruning rule. The new approximation procedure, MCLL, is specified in Algorithm 3.3.

### 3.3.5 Optimization

Due to the noisiness of  $L$  over the bandwidth space, its gradient is of little utility. Thus, we adopt a scheme similar to that of Gray and Moore [73]: evaluate the likelihood at all combinations of  $h_1 \in \{10^{-4}, 10^{-3}, \dots, 10^2\}$  and  $h_2 \in \{10^{-4}, 10^{-3}, \dots, 10^2\}$ . Given whitened data, this allows us to cover all orders of magnitude within a range that, in practice, is more than wide enough to cover the optimal bandwidth pair. Evaluating over this grid allows us to gauge the composite performance of our likelihood approximations over the full range of bandwidths, from near-optimal to far-from-optimal. For applications, one might wish to evaluate on a grid with somewhat finer resolution, e.g., at intervals  $\{0.25, 0.5, 0.75, 1\}$  within each order of magnitude.

## 3.4 Experiments

We present results from two sets of experiments. The first set is designed to test the efficiency and accuracy of our likelihood approximation algorithms across a broad array of datasets. In order to gauge the speedup and error performance of our methods, we restrict ourselves to medium-sized datasets (17,000–60,000 points) so that the baseline exact evaluations can be done in reasonable time. In the second set of experiments, we test the scalability of our Monte Carlo procedure by applying it to datasets on the order of hundreds of thousands to millions of points. Datasets of this size represent the largest reported application of KCDE to date, by several orders of magnitude.

We note several details about the experimental methodology. First, all data was whitened (mean subtracted, divided by standard deviation) on a per-dimension basis. This allows us to search the same bandwidth range for all datasets, while enabling  $h_2$  to effectively provide a variable bandwidth  $h_2\sigma_d$  for each dimension  $d$  of  $x$ . We used the asymptotically optimal Epanechnikov kernel in all cases (see [134]).

The nature of the cross-validated likelihood score  $L$  is such that it often evaluates

to  $-\infty$ . The MCLL approximation occasionally returns a finite value for bandwidth pairs whose true score is  $-\infty$ . This happens in cases where a few points are far enough from the others that they receive no cross-validated probability density. It is not problematic in practice, and, in fact, one could argue that this is a “feature” that makes MCLL more robust than exact cross-validated likelihood. In order not to skew the picture of accuracy, however, such instances were left out of the error averages. Both speedup and error measurements for Monte Carlo approximation are averaged first across five invocations per evaluated bandwidth pair, then across all bandwidth pair averages. For deterministic approximation, we average only across all bandwidth pairs.

In order to compensate for the overconservatism of our approximation framework and to put the two approximation methods on equal ground in terms of error performance, the error tolerances for the deterministic and Monte Carlo approximations were set to  $\epsilon = 700$  and  $\epsilon = 1$ , respectively. This was chosen to consistently give actual errors of no more than 0.1. The additional parameter settings for the Monte Carlo algorithm were:  $m = 25$  (sample size);  $B = 10$  (resampling replications);  $z = 1.5$  (number of standard errors for confidence interval).

All experiments were run sequentially on a two-processor Intel Xeon 3.8GHz hyperthreading machine with 8GB memory.

### 3.4.1 Error Control and Speedup on Medium-Scale Datasets

In the first set of experiments, we ran the full optimization procedure described in Section 3.3.5 on seven medium-sized datasets ranging from 17,605 to 61,634 points in 3–16 dimensions. For each bandwidth pair evaluated during the optimization, we recorded the evaluation time and score for 1) exact evaluation, 2) deterministic dual-tree approximation (Algorithm 3.2), and 3) Monte Carlo dual-tree approximation (Algorithm 3.3). For both deterministic and Monte Carlo approximation, we

**Table 1:** Results for deterministic dual-tree likelihood approximation over a range of medium-sized datasets.

dataset	$n$	$d$	exact time (min)	det. time (min)	speedup	error
sp500	17605	6	296.2	30.9	9.6	0.03
census	20640	9	203.9	136.4	1.5	0.02
spectral	46420	5	818.8	374.8	2.2	0.02
sdss50k	50000	3	621.5	110.0	5.7	0.04
galaxy50k	50000	3	424.7	99.2	4.3	0.02
quasar50k	50000	4	1337.2	384.7	3.5	0.009
corel	61634	16	2533.0	1380.6	1.8	0.032

**Table 2:** Results for Monte Carlo dual-tree likelihood approximation over a range of medium-sized datasets.

dataset	$n$	$d$	exact time (min)	MC time (sec)	speedup	error
sp500	17605	6	296.2	2.2	8,265	0.05
census	20640	9	203.9	2.6	4,745	0.01
spectral	46420	5	818.8	5.7	8,606	0.01
sdss50k	50000	3	621.5	6.5	5,746	0.05
galaxy50k	50000	3	424.7	4.1	6,165	0.05
quasar50k	50000	4	1337.2	4.9	16,233	0.0006
corel	61634	16	2533.0	13.1	11,598	0.009

compare the approximate likelihood scores to the exact scores and report the average error across all evaluated bandwidths. We also report the speedup in terms of total evaluation time over all bandwidths, i.e.,  $speedup_{approx} = time_{exact}/time_{approx}$ . Tables 1–2 give the numerical results.

Several patterns are apparent in these results. First, both the deterministic and Monte Carlo dual-tree methods give speedup over the exact method while keeping error small. Speedups for the deterministic algorithm, however, are less than 10 in all cases, while the addition of Monte Carlo yields speedups from the mid-thousands to just over ten thousand. Adding Monte Carlo to the dual-tree methodology is therefore seen to give a 2–3 order-of-magnitude boost over the deterministic approach. Again,

the reason for this is that Monte Carlo bounds are much tighter than the worst-case deterministic bounds, leading to higher prunes and far less overall work being performed.

The cost of increased speedup for Monte Carlo is the loss of the strict error guarantee provided by the deterministic approximation. As we can see, however, error is still well controlled by the Monte Carlo algorithm, so this is a small sacrifice. Average absolute errors are below 0.1 in all cases, which represents a typical relative error of less than 5%.

Another important pattern is the behavior of speedup as datasets increase in size. In the Monte Carlo case, speedup clearly increases with dataset size, and this is further validated in the scalability experiments of the next section. In the deterministic case, the pattern is less clear, but may show a slight increasing trend. This is another strong reason to prefer the Monte Carlo algorithm for the large data case, as its speedup level is more likely to be able to keep up with the  $O(n^2)$  growth in computational cost of the exact algorithm.

All together, these medium-scale results indicate that both the deterministic and Monte Carlo dual-tree algorithms provide high-quality approximations, but the Monte Carlo approach is much faster. We therefore use the Monte Carlo algorithm exclusively as we examine the scalability of our likelihood approximation scheme.

### 3.4.2 Scalability on Large Datasets

We now address the large-data regime for which fast approximation is most important. In particular, we give results for our three largest datasets, each of which contains on the order of  $10^5$ – $10^6$  points. Because exact likelihood computation is too expensive to compute on datasets of this size, we instead extrapolate the exact runtime by fitting a quadratic curve to the exact runtimes of optimization on subsamples of sizes  $\{10, 25, 50, 75\} \times 10^4$ . We then run the optimization using Monte Carlo dual-tree

**Table 3:** Results for Monte Carlo dual-tree likelihood approximation on a set of large-scale datasets.

<b>dataset</b>	$n$	$d$	<b>exact time</b>	<b>MC time</b>	<b>speedup</b>
sdss	389,353	3	26.4 days	33.7 sec	67,611
galaxy1M	1,000,000	3	197.4 days	1.7 min	169,572
quasar1M	1,000,000	4	1.2 years	2.1 min	286,564

approximation and report timing results. Table 3 summarizes the performance.

We see in these results a continuation of the pattern of substantially increasing speedup as dataset size increases. Speedups range from around 67,000 to over 286,000. In absolute terms, the reductions are quite significant, e.g., from 1.2 years to 2.1 minutes in the largest case. Relative to the medium-sized datasets, the Monte Carlo runtimes went from several seconds to several minutes, while the exact runtimes went from hundreds of minutes to tens and hundreds of days. This demonstrates excellent scalability. Our application of KCDE to datasets of size  $10^6$  is larger than any previously published application of KCDE by three orders of magnitude [80]. Combined with the good error control demonstrated in the previous section, these results would seem to make MCLL the algorithm of choice for KCDE likelihood optimization on large datasets.

### **3.5 Conclusion**

We have presented a fast, scalable, approximate method for learning kernel conditional density estimates. This method injects Monte Carlo sampling into the dual-tree framework that has previously been used to produce state-of-the-art scalable learning for other kernel estimators. Our Monte Carlo method is powerful because it allows KCDE to be applied to datasets and applications for which it would never have been considered before. Because KCDE is such a fundamental and versatile method, having a fast, scalable version opens the door to the asking of new questions, new forms of visualization and analysis, the development of new methods that would previously

have been dismissed as too costly, and so on.

Experimentally, we have shown that our Monte Carlo dual-tree method controls error, produces speedups ranging from  $10^3$ – $10^5$ , and scales to datasets on the order of at least one million points. Absolute computation time reductions were as large as from 1.2 years to 2.1 minutes. These are very significant speedups, and represent an advance of at least several orders of magnitude in both speed and addressable dataset size over the prior state of the art.

The major drawback to this Monte Carlo dual-tree algorithm is the loss of strict error guarantees relative to the deterministic dual-tree method. Our next work, therefore, seeks to address this shortcoming. It does so by flipping from the “Monte Carlo in trees” approach used here to a “trees in Monte Carlo” approach that puts the approximation on a fully Monte Carlo footing while still leveraging the power of spatial partitioning. As we will see, this both increases algorithmic acceleration and further enables us to broaden the scope of applicability beyond KCDE to include fast learning procedures for all kernel estimators.

## CHAPTER IV

### TREES IN MONTE CARLO: FAST GENERAL KERNEL ESTIMATORS

We now go beyond the methodology of Chapter 3 in two key ways. First, we move to a fully Monte Carlo framework, within which we will continue to leverage the power of spatial partitioning trees [81]. This is the idea of injecting trees into Monte Carlo, and it enables rigorous relative error control and efficient small-sample estimation. Second, we generalize the class of computations to which the methodology can be applied. While the fast KCDE training method of Chapter 3 was derived for approximating a specific objective function, the algorithm we describe here can be used to approximate any computation within a generalized class of nested-summative forms. The approximation objective from Chapter 3 is subsumed within this class, as are the training objectives of the other standard kernel estimators such as kernel regression and kernel density estimation. While the kernel estimators are our main focus, the nested-summative class of computations includes many others found throughout machine learning, optimization, and related fields. We show conditions for high-probability error control, give a sample complexity bound that is independent of dataset size (implying asymptotically infinite speedups), and demonstrate scalability with speedups as high as  $10^6$  on datasets with as many as tens of millions of points. In absolute terms, such speedups represent reductions from years to minutes of computation time. These results are several orders of magnitude better than the previous state of the art, providing a strong showcase for the power of the Multi-Tree Monte Carlo methodology.

## 4.1 *Introduction*

Many machine learning methods have computational bottlenecks in the form of nested summations that become impractical in applications involving large datasets. We are particularly motivated by the nonparametric kernel estimators (e.g., kernel density estimation), but a variety of other methods require computations of similar form. In this work we formalize a general class of nested-summative computational forms and present a new recursive Monte Carlo method for approximating any computation of nested-summative form with rigorous relative error control. Key to the efficiency of our method is the use of tree-based data stratification, i.e., sampling in trees. We derive error guarantees and sample complexity bounds, with the intriguing result that sample complexity depends not on dataset size but on statistical features of the sample mean that can be controlled through stratification. We also present experiments that validate these theoretical results and demonstrate several-order-of-magnitude speedup over the prior state of the art.

Previous approaches to algorithmic acceleration of this kind fall into roughly two groups: 1) methods that run non-accelerated algorithms on subsets of the data, typically without error bounds, and 2) multi-tree methods with automatic, deterministic error control. The former are of less interest due to the lack of error control, while the latter are good when deterministic error control is required, but have built-in overconservatism that limits speedup, and are difficult to extend to new problems. Our Monte Carlo approach offers much larger speedup and a generality that makes it simpler to adapt to new problems, while retaining strong automatic error control to within user-specified tolerances. While there are non-summative computational forms to which the standard multi-tree methodology is applicable and our Monte Carlo method is not, our method appears to give greater speedup by several orders of magnitude on problems where both methods can be used.

In summary, this work makes the following contributions: formulation of the class

of generalized nested-summative computations; a new recursive Monte Carlo algorithm with automatic error control for nested-summative forms; sample complexity bounds showing no explicit dependence on dataset size; a variance-driven, tree-based stratification scheme for efficient sampling from datasets; application of stratification scheme for efficient small-sample approximation; application of general algorithm to fast training of kernel estimators; empirical speedups as high as  $10^6$  on  $10^7$  points.

## 4.2 Problem Definition and Previous Work

We first illustrate the problem class by giving expressions for the cross-validated squared-error scores used to optimize bandwidths in kernel regression (KR) and kernel density estimation (KDE):

$$S_{KR} = \frac{1}{n} \sum_i \left( y_i - \frac{\sum_{j \neq i} K_h(\|x_i - x_j\|) y_j}{\sum_{j \neq i} K_h(\|x_i - x_j\|)} \right)^2$$

$$S_{KDE} = \frac{1}{n^2} \sum_i \sum_j \int K_h(\|x - x_i\|) K_h(\|x - x_j\|) dx - \frac{2}{n(n-1)} \sum_i \sum_{j \neq i} K_h(\|x_i - x_j\|) .$$

These nested sums have unscalable  $O(n^2)$  and  $O(n^3)$  computation times that make them impractical on large datasets. We seek to develop a fast, error-controlled approximation scheme that will make these computations scalable. Further, we want our approximation scheme to be general enough to include all the nested-summative forms encountered in kernel estimators and related methods. We therefore begin by formalizing the class of computations we will address.

**Definition 1.** *A nested-summative-form computation is one that fits the following inductive pattern:*

$$B(\mathcal{X}_c) \rightarrow \sum_{i \in I(\mathcal{X}_c)} f(\mathcal{X}_c, \mathcal{X}_i) \tag{16}$$

$$G(\mathcal{X}_c) \rightarrow B(\mathcal{X}_c) \mid \sum_{i \in I(\mathcal{X}_c)} f(\mathcal{X}_c, \mathcal{X}_i, G_1(\mathcal{X}_c, \mathcal{X}_i), G_2(\mathcal{X}_c, \mathcal{X}_i), \dots) , \tag{17}$$

where  $f$  is a differentiable function.

$$\begin{aligned}
S_{KR} &= \frac{1}{n} \sum_i \left( y_i - \frac{\sum_{j \neq i} K_h(\|x_i - x_j\|) y_j}{\sum_{j \neq i} K_h(\|x_i - x_j\|)} \right)^2 \\
&= \sum_{i \in I} f(\mathcal{X}_i, G_1(\mathcal{X}_i), G_2(\mathcal{X}_i)) \\
I &= \{1 \dots n\} \\
f(\mathcal{X}_i, G_1, G_2) &= \frac{1}{n} \left( y_i - \frac{G_1(\mathcal{X}_i)}{G_2(\mathcal{X}_i)} \right)^2 \\
&\quad \swarrow \qquad \searrow \\
&\quad \sum_{j \neq i} K_h(x_i, x_j) y_j \qquad \sum_{j \neq i} K_h(x_i, x_j)
\end{aligned}$$

**Figure 7:** Mapping the kernel regression score  $S_{KR}$  into the general case  $G$  of the nested-summative formalism.

$B$  represents the base case, in which a tuple of constant arguments  $\mathcal{X}_c$  may be specified and a tuple of variable arguments  $\mathcal{X}_i$  is indexed by a set  $I$ , which may be a function of  $\mathcal{X}_c$ . The arguments  $\mathcal{X}_c$  and  $\mathcal{X}_i$  pass through a function  $f$  to generate the terms of the summation. For instance, the mean of a univariate size- $n$  dataset is of type  $B$ , with  $\mathcal{X}_c = \langle \rangle$ ,  $I(\mathcal{X}_c) = \{1, \dots, n\}$ ,  $\mathcal{X}_i = \langle x_i \rangle$ , and  $f(\mathcal{X}_c, \mathcal{X}_i) = \frac{x_i}{n}$ . Note that  $|I|$  is the number of terms in a summation of type  $B$ , and therefore  $O(|I|)$  represents the base time complexity (i.e., the cost of brute-force evaluation). Whenever  $I$  indexes all  $k$ -tuples or leave-one-out  $k$ -tuples over the dataset, the base complexity is  $O(n^k)$ , where  $n$  is the size of the dataset.

The general case  $G$  is an inductive form, and maps to either: 1) the base case  $B$ , or 2) a sum nearly identical to the base case, but where the summand function  $f$  is allowed to have additional arguments which are themselves summative forms of type  $G$ . A simple example of this is the cross-validated mean squared error score from kernel regression (i.e.,  $S_{KR}$  listed above). Figure 7 illustrates how  $S_{KR}$  maps to the general form  $G$ . Note that the variable argument  $\mathcal{X}_i$  of the outer summation becomes the constant argument  $\mathcal{X}_c$  of the inner leave-one-out summations, and that the inner summations have an index function  $I$  that leaves out the  $i$ th index value.

The base complexity of a type- $G$  computation is  $|I|$  multiplied by the maximum

base complexity among the nested instances. For instance, in  $S_{KR}$  we have  $I = \{1, \dots, n\}$ , so  $|I| = n$ , while the inner summations  $G_1$  and  $G_2$  are  $O(n)$ , for an overall base complexity of  $O(n^2)$ .

#### 4.2.1 Previous Work

Though the present work is the first to define the general nested-summative class of computational forms, other works have devised approximations for special cases and related forms. Previous work has fallen into roughly two groups. First are methods where data is simply subsampled before running a non-accelerated algorithm. Stochastic gradient descent and its variants (e.g., [131]) are prototypical here. While such approaches can have asymptotic convergence, there are no error guarantees for finite sample sizes. This is not show-stopping in practice, but the lack of quality assurance is a critical shortcoming. Our approach also exploits the speedup that comes from sampling, but provides a rigorous relative error guarantee and is able to automatically determine the necessary sample size to provide that guarantee.

The other main class of acceleration methods consists of those employing “higher order divide and conquer” or multi-tree techniques that give either exact answers or approximations with automatic, deterministic error control [71, 66, 73, 67, 65, 68, 69, 98, 26, 94, 93, 70, 92, 97, 117, 132, 139, 118]. The idea is that the user specifies an error tolerance  $\epsilon$ , and the algorithm automatically performs the least work it can while achieving an approximation of that quality. Multi-tree methods apply to a broad class of “generalized  $n$ -body problems” (GNPs), and feature the use of multiple spatial partitioning structures such as  $kd$ -trees or ball trees to decompose and reuse portions of computational work. While a full formal definition of GNPs has yet to be published, it is clear that nested-summative forms are closely related and have at least partial overlap. One important case of GNP that is *not* covered by nested-summative forms is the for-all construct, e.g., individually predicting a regression

target or evaluating a density for each point in a query set.

The standard multi-tree methodology has three significant drawbacks. First, although it gives deterministic error bounds, the bounds are usually quite loose, resulting in overconservatism that prevents aggressive approximation that could give greater speed. Second, creating a new multi-tree method to accelerate a given algorithm requires complex custom derivation of error bounds and pruning rules. Third, the standard multi-tree approach is conjectured to reduce  $O(n^k)$  computations at best to  $O(n^{\log k})$ . This still leaves an unscalable computation for  $k$  as small as 4.

In [80], the first of these concerns began to be addressed by employing sample-based bounds within a deterministic dual-tree error propagation framework. The essential idea in that case was injecting Monte Carlo into trees. The present work builds on that idea by moving to a fully Monte Carlo scheme where multiple trees are used for variance-reducing stratification. The conceit here is injecting trees into Monte Carlo. Error is rigorously, automatically controlled using sample variance, allowing the Monte Carlo approach to make aggressive approximations and avoid the overconservatism of deterministic multi-tree methods. This yields greater speedups by several orders of magnitude. Further, our Monte Carlo approach handles the class of nested-summative forms in full generality, making it easier to specialize to new problems. Lastly, the computational complexity of our method is not directly dependent on dataset size, which means it can address higher degrees of nesting that would make the standard multi-tree approach intractable. The price we pay for these improvements is that our Monte Carlo error bounds are probabilistic, though the bound probability is controlled as a parameter to the algorithm. For applications where the data is large and/or minor stochasticity in the approximated output is tolerable, this is a good sacrifice.

### 4.3 Algorithmic Development

In deriving the general algorithm for nested-summative forms, we need to develop three main ideas: 1) recursive Monte Carlo estimation, 2) delta-method chaining, and 3) data-tree-driven stratified sampling. Recursive Monte Carlo allows us to handle the nested aspect of nested-summative computations. Delta-method chaining enables the confidence intervals we need to guarantee error bounds. Data-tree stratification makes our method effective with very small sample sizes, generating the speedups required to handle large datasets. The combination of these elements allows us to give a general approximation algorithm that handles all nested-summative forms, approximating them to a user-specified relative error tolerance with high efficiency.

#### 4.3.1 Central Limit Conditions

Before deriving our algorithm, we need to define and justify the conditions under which it will be proven to work. We begin with a statement of the Central Limit Theorem (CLT).

**Theorem 2.** Central Limit Theorem. *Let  $P$  be a probability distribution over scalar values  $x$  with expectation  $\mu$  and variance  $\sigma^2$ . Let  $\hat{\mu}$  be the mean of  $m$  samples drawn from  $P$ , and let  $\hat{\sigma}^2$  be the sample variance. The following convergence results hold as  $m \rightarrow \infty$ :*

$$\hat{\mu} \rightsquigarrow N(\mu, \sigma^2/m) \tag{18}$$

$$\hat{\mu} \rightsquigarrow N(\mu, \hat{\sigma}^2/m) \tag{19}$$

where  $N(a, b)$  is the Gaussian distribution with mean  $a$  and variance  $b$ .

Because the CLT is an asymptotic result, the question arises: how well does it describe the small-sample case? A widely accepted statistical rule of thumb asserts that 30 or more samples are usually enough to put a sample mean into the asymptotic regime. A more exact characterization, however, is found in the Berry-Esséen

theorem. While the Berry-Esséen theorem can be stated more generally, we give here a specialized version tailored to the CLT statement of Theorem 2.

**Theorem 3.** Berry-Esséen Theorem. *Let  $\hat{\mu}$  be the sample mean of  $m$  samples drawn from the distribution  $P$ , and let  $\mu$ ,  $\sigma^2$ , and  $\rho$  be the mean, variance, and third absolute central moment of  $P$ . Let  $F_m(x)$  be the cumulative distribution function (cdf) of  $\hat{\mu}$ , and let  $\Phi(x; \mu, \sigma^2)$  be the cdf of the Gaussian with mean  $\mu$  and variance  $\sigma^2$ . Then there exists a positive constant  $C$  such that for all values of  $\hat{\mu}$  and  $m$ :*

$$|F_m(\hat{\mu}) - \Phi(\hat{\mu}; \mu, \sigma^2)| \leq \frac{C\rho}{\sigma^3\sqrt{m}}. \quad (20)$$

The Berry-Esséen theorem thus provides a uniform convergence bound for the *actual* distribution of a sample mean  $\hat{\mu}$  relative to its *asymptotic* distribution  $N(\mu, \sigma^2/m)$ . Specifically, the bound on the difference between the two cdfs at any point is proportional to the sample-mean absolute skewness  $\frac{\rho}{\sigma^3\sqrt{m}}$ , which goes down as  $1/\sqrt{m}$ . The constant  $C$  is known to be no greater than 0.7655 [133]. Thus, the lower the absolute skewness of the sample mean distribution, the fewer the samples required for  $\hat{\mu}$  to reach its asymptotic distribution. Given a sufficiently high sample size  $m$ , the skewness factor is overwhelmed and the right-hand side of the Berry-Esséen bound becomes effectively zero. In this case, the sample mean  $\hat{\mu}$  can be considered to be in its asymptotic regime. We formalize this idea using a pair of definitions that will prove useful in establishing the conditions for our algorithms' error guarantees. Note that these definitions are given in terms of a general estimator  $\hat{\theta}$  for which a CLT-type convergence holds; the sample mean is included as a special case.

**Definition 2.**  $\epsilon$ -asymptotic. *Let  $\hat{\theta}_m$  be a sample estimator from  $m$  samples, let  $\hat{\sigma}^2/m$  be a sample estimate of  $V[\hat{\theta}_m]$ , and let  $\theta$  be the true value being estimated. If  $\hat{\theta}_m$  satisfies a central limit theorem  $\hat{\theta}_m \rightsquigarrow N(\theta, \hat{\sigma}^2/m)$  as  $m \rightarrow \infty$ , we say that  $\hat{\theta}_m$  is  $\epsilon$ -asymptotic if  $|F_{\hat{\theta}_m}(t) - \Phi(t; \theta, \hat{\sigma}^2/m)| \leq \epsilon$  holds for all values  $t$ .*

This definition describes an  $\epsilon$ -closeness between the actual cdf of the sampling distribution and its asymptotic cdf. Our particular interest is the case where the distribution of an estimator is not *discernibly* different from its asymptotic distribution, in which case we can legitimately invoke asymptotic confidence bounds. While the precision required for discernibility will vary by application, machine precision at least presents a finite upper bound. Thus, there is always an  $\epsilon > 0$  for which  $\epsilon$ -asymptotic behavior is indistinguishable from true asymptotic behavior. Further, the Berry-Essén theorem suggests that a CLT-governed estimator can be made  $\epsilon$ -asymptotic with finite  $m$  on the order of  $C\rho/\epsilon\sigma^3$ . This leads us to conclude that the  $\epsilon$ -asymptotic property required for invoking CLT-based confidence bounds is achievable with finitely many samples. This notion is encapsulated in the following definition.

**Definition 3.** *CLT-sufficient. Let  $\hat{\theta}$  be a sample estimator from  $m$  samples satisfying a central limit theorem  $\hat{\theta} \rightsquigarrow N(\theta, \hat{\sigma}^2/m)$ . If  $m$  is large enough that  $\hat{\theta}$  is  $\epsilon$ -asymptotic for  $\epsilon$  small enough that asymptotic confidence bounds hold to a required degree of precision, then we say that  $m$  is CLT-sufficient for  $\hat{\theta}$  at the required degree of precision.*

This notion of a CLT-sufficient sample size will characterize the conditions under which our approximation algorithms can guarantee relative error bounds. In our analysis, we assume the required level of precision is that which enables the relative error bounds to hold to within machine precision. The question of whether we can find *reasonable* CLT-sufficient sample sizes thus becomes an empirical one. As we will see through our experiments, the answer to this question is yes.

### 4.3.2 Base Case: Flat Monte Carlo

Before handling the general case of  $G$ -type summations, we first derive a Monte Carlo approximation for the base case of  $B$ -type flat summations (Equation 16). The basic algorithm for this case is similar to standard Monte Carlo approximation as in [123] or [61], but the development of conditions for error control is our own. With the

base algorithm in place, we proceed to derive novel sample complexity bounds and then extend the  $B$ -type algorithm to handle nested  $G$ -type forms. As a final step, we inject tree-stratified sampling for a large efficiency boost. The combination of all these elements will yield an efficient, small-sample approximation algorithm for the entire class of nested-summative forms, while generating insights into the dependence of computational complexity on sample statistics and how tree-based methods can improve those statistics.

To start, note that in a  $B$ -type summation  $S(\mathcal{X}_c) = \sum_{i \in I(\mathcal{X}_c)} f(\mathcal{X}_c, \mathcal{X}_i)$ , the summation can be written as  $S(\mathcal{X}_c) = nE[f_i] = n\mu_f$ , where  $n = |I(\mathcal{X}_c)|$  and the expectation is taken over a discrete distribution  $P_f$  that puts mass  $\frac{1}{n}$  on each term  $f_i = f(\mathcal{X}_c, \mathcal{X}_i)$ . Our goal is to produce an estimate  $\widehat{S}$  that has low relative error with high probability. More precisely, for a specified  $\epsilon$  and  $\delta$ , we want  $|\widehat{S} - S| \leq \epsilon|S|$  to hold with probability at least  $1 - \delta$ . This error bound on  $\widehat{S}$  is achieved if we let  $\widehat{S} = n\widehat{\mu}_f$  for an estimate  $\widehat{\mu}_f$  of the mean  $\mu_f$  that satisfies the analogous relative error bound  $|\widehat{\mu}_f - \mu_f| \leq \epsilon|\mu_f|$ .

Let  $\widehat{\mu}_f$  be the mean of  $m$  samples taken from  $P_f$ , and let  $m$  be CLT-sufficient (Definition 3). We therefore have  $\widehat{\mu}_f \sim N(\mu_f, \widehat{\sigma}_f^2/m)$ , from which we can construct the normal  $1 - \delta$  confidence interval:

$$|\widehat{\mu}_f - \mu_f| \leq z_{\delta/2} \widehat{\sigma}_f / \sqrt{m}, \quad (21)$$

where  $z_{\delta/2}$  is the number of standard deviations on either side of the mean required to give  $1 - \delta$  coverage under the normal distribution. Given this bound on  $|\widehat{\mu}_f - \mu_f|$ , our relative error condition  $|\widehat{\mu}_f - \mu_f| \leq \epsilon|\mu_f|$  is implied by  $z_{\delta/2} \widehat{\sigma}_f / \sqrt{m} \leq \epsilon|\mu_f|$ . The bound also implies a lower limit for  $|\mu_f|$ :  $|\mu_f| \geq |\widehat{\mu}_f| - z_{\delta/2} \widehat{\sigma}_f / \sqrt{m}$ . Combining these, we can ensure our target relative error by requiring that  $z_{\delta/2} \widehat{\sigma}_f / \sqrt{m} \leq \epsilon(|\widehat{\mu}_f| - z_{\delta/2} \widehat{\sigma}_f / \sqrt{m})$ , which rearranges to:

$$m \geq z_{\delta/2}^2 \frac{(1 + \epsilon)^2 \widehat{\sigma}_f^2}{\epsilon^2 \widehat{\mu}_f^2}. \quad (22)$$

Inequality 22 is an empirically testable condition that depends only on sample statistics. The satisfaction of this condition guarantees that the estimate  $\widehat{S} = n\widehat{\mu}_f$  satisfies our target relative error bound with probability  $1 - \delta$ , given the CLT-sufficiency of the sample size  $m$ . This suggests an iterative sampling procedure in which we start with a number of samples  $m_{min}$  large enough to ensure CLT-sufficiency, and then draw additional samples until Inequality 22 is met. This is the FLATMC procedure listed in Algorithm 4.1. Note that the algorithm uses the gap between the current sample size  $m$  and the threshold  $z_{\delta/2}^2 \frac{(1+\epsilon)^2 \hat{\sigma}_f^2}{\epsilon^2 \hat{\mu}_f^2}$  to determine how many additional samples to draw after each check of the Inequality 22 condition. We state the error guarantee for FLATMC as Theorem 4.

**Theorem 4.** *Given a summation  $S$ ,  $\epsilon \in [0, 1]$ ,  $\delta \in [0, 1]$ , and a CLT-sufficient value of  $m_{min}$ , with probability at least  $1 - \delta$  the algorithm FLATMC returns an approximation  $\widehat{S}$  satisfying  $|\widehat{S} - S| \leq \epsilon|S|$ .*

*Proof.* We have already established that, given a CLT-sufficient sample size  $m$ , Inequality 22 implies that  $|\widehat{S} - S| \leq \epsilon|S|$  holds with probability at least  $1 - \delta$ . Algorithm 4.1 starts with the CLT-sufficient sample size  $m_{min}$  and only increases it, so the terminal sample size is always CLT-sufficient. The algorithm alternates between adding samples and checking the error condition. We need to show that the multiple intermediate error checks do not induce a failure rate of more than  $\delta$ , where failure is defined as returning an approximation  $\widehat{S}$  not satisfying  $|\widehat{S} - S| \leq \epsilon|S|$ .

The key to bounding the failure rate is to note that only the terminal error test matters. This is because intermediate error tests only ever tell the algorithm to keep sampling, and this is never a failure. Thus, only the single terminal test can cause failure, and we know that a single test fails with probability no greater than  $\delta$  (property of Inequality 22). The algorithm therefore succeeds in satisfying the error bound  $|\widehat{S} - S| \leq \epsilon|S|$  with probability at least  $1 - \delta$ .  $\square$

---

**Algorithm 4.1** Iterative Monte Carlo approximation for flat summations.

---

FLATMC

**Input:** type- $B$  summation  $S$ ; constant  $\mathcal{X}_c$ ;  $\epsilon, \delta \in [0, 1]$ ; CLT-sufficient  $m_{min}$

**Output:**  $\hat{S}$  s.t.  $|\hat{S} - S| \leq \epsilon S$  with probability at least  $1 - \delta$

1.  $samples \leftarrow \emptyset, m_{needed} \leftarrow m_{min}$
2. **repeat**
  - (a)  $ADDSAMPLES(samples, m_{needed}, S, \mathcal{X}_c)$
  - (b)  $m, \hat{\mu}_f, \hat{\sigma}_f^2 \leftarrow CALCSTATS(samples)$
  - (c)  $m_{thresh} \leftarrow z_{\delta/2}^2(1 + \epsilon)^2 \hat{\sigma}_f^2 / \epsilon^2 \hat{\mu}_f^2$
  - (d)  $m_{needed} \leftarrow m_{thresh} - m$
3. **until**  $m \geq m_{thresh}$
4. **return**  $|S.I(\mathcal{X}_c)|\hat{\mu}_f$

ADDSAMPLES

**Input:** sample list  $samples$ ; samples to add  $m$ ; type- $B$  summation  $S$ ; constant  $\mathcal{X}_c$

**Output:**  $m$  samples drawn uniformly from the terms of  $S$ , stored in  $samples$

1. **for**  $i = 1$  to  $m$ 
  - (a)  $\mathcal{X}_i \leftarrow rand(S.I(\mathcal{X}_c))$
  - (b)  $APPEND(S.f(\mathcal{X}_c, \mathcal{X}_i), samples)$

CALCSTATS

**Input:** sample list  $samples$

**Output:** sample count  $m$ , sample mean  $\hat{\mu}$ , and sample variance  $\hat{\sigma}^2$

1.  $m \leftarrow COUNT(samples), \hat{\mu} \leftarrow AVG(samples), \hat{\sigma}^2 \leftarrow VAR(samples)$
  2. **return**  $m, \hat{\mu}, \hat{\sigma}^2$
- 

**Sample Complexity.** Because we are interested in *fast* approximations, Algorithm 4.1 is only useful if it terminates with  $m$  significantly smaller than the number of terms in the full summation. Equation 22 gives an empirical test indicating when  $m$  is large enough for sampling to terminate; we now provide an upper bound, in terms of the distributional properties of the summation terms  $f_i$ , for the value of  $m$  at which Equation 22 will be satisfied. Note that for this theorem we extend the notion

of CLT-sufficiency to cover both  $\hat{\mu}$  and  $\hat{\sigma}^2$  being in their asymptotic distributions.

**Theorem 5.** *Given a summation  $S$ ,  $\epsilon \in [0, 1]$ ,  $\delta \in [0, 1]$ , and a value  $m_{min}$  that is CLT-sufficient for both  $\hat{\mu}_f$  and  $\hat{\sigma}_f^2$ , with probability at least  $1 - 2\delta$  the algorithm FLATMC will terminate with sample size  $m \leq O\left(\frac{\sigma_f^2}{\mu_f^2} + \frac{\sigma_f}{|\mu_f|} \sqrt{\frac{\mu_{4f}}{\sigma_f^4} - 1}\right)$ .*

*Proof.* The termination condition of FLATMC (Inequality 22) is driven by  $\hat{\sigma}_f^2/\hat{\mu}_f^2$ , so we proceed by bounding this ratio. First, the CLT implies that with probability  $1 - \delta$  we have the following lower bound on the absolute value of the sample mean:

$$|\hat{\mu}_f| \geq |\mu_f| - z_{\delta/2}\sigma_f/\sqrt{m}. \quad (23)$$

Next,  $\hat{\sigma}_f^2$  has the asymptotic distribution  $N(\sigma_f^2, (\mu_{4f} - \sigma_f^4)/m)$ , where  $\mu_{4f}$  is the fourth central moment of the distribution  $P_f$  over summation term. We can therefore apply the delta method to infer that  $\hat{\sigma}_f$  is distributed as  $N(\sigma_f, (\mu_{4f} - \sigma_f^4)/4\sigma_f^2m)$ . Using the normal-based confidence interval (justified by the CLT-sufficiency of  $m_{min}$ ), this gives the following  $1 - \delta$  upper bound for the sample standard deviation:

$$\hat{\sigma}_f \leq \sigma_f + z_{\delta/2}\sqrt{(\mu_{4f} - \sigma_f^4)/(2\sigma_f\sqrt{m})}. \quad (24)$$

We now combine the upper bound on  $\hat{\sigma}_f$  with the lower bound on  $\hat{\mu}_f$ . Since we only know that each bound individually covers at least a  $1 - \delta$  fraction of outcomes, we can only guarantee they will jointly hold with probability at least  $1 - 2\delta$ . We thus have the following bound with probability at least  $1 - 2\delta$ :

$$\frac{\hat{\sigma}_f}{|\hat{\mu}_f|} \leq \frac{\sigma_f + z_{\delta/2}\frac{\sqrt{\mu_{4f} - \sigma_f^4}}{2\sigma_f\sqrt{m}}}{|\mu_f| - z_{\delta/2}\frac{\sigma_f}{\sqrt{m}}}. \quad (25)$$

Substituting this into the right side of the termination condition (Inequality 22) and solving for  $m$  shows that, with probability at least  $1 - 2\delta$ , the algorithm will terminate with  $m$  no larger than:

$$\frac{z_{\delta/2}^2(1+2\epsilon)^2}{2} \frac{\sigma_f}{\epsilon^2} \frac{\sigma_f}{|\mu_f|} \left[ \frac{\sigma_f}{|\mu_f|} + \frac{\epsilon(1+\epsilon)}{(1+2\epsilon)^2} \sqrt{\frac{\mu_{4f}}{\sigma_f^4} - 1} + \sqrt{\frac{\sigma_f}{|\mu_f|}} \sqrt{\frac{\sigma_f}{|\mu_f|} + \frac{2\epsilon(1+\epsilon)}{(1+2\epsilon)^2} \sqrt{\frac{\mu_{4f}}{\sigma_f^4} - 1}} \right], \quad (26)$$

which is bounded by  $O\left(\frac{\sigma_f^2}{\mu_f^2} + \frac{\sigma_f}{|\mu_f|} \sqrt{\frac{\mu_{4f}}{\sigma_f^4} - 1}\right)$ . □

Three aspects of this bound are salient. First, it liberates computation time from dataset size. This is because the sample complexity depends only on the *distributional features* ( $\sigma_f^2$ ,  $\mu_f$ , and  $\mu_{4f}$ ) of the summation terms, and not on the *number* of terms. For i.i.d. datasets in particular, these distributional features are convergent, which means the sample/computational complexity converges to a constant while speedup is unbounded as the dataset size goes to infinity.

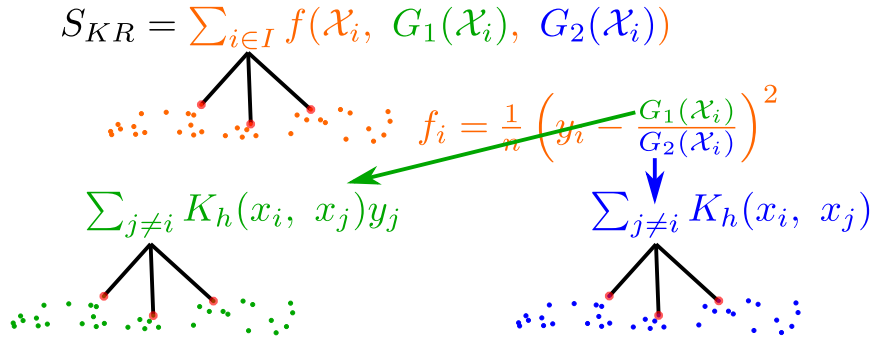
Second, the bound has sensible dependence on  $\sigma_f/|\mu_f|$  and  $\mu_{4f}/\sigma_f^4$ . The former is a standard dispersion measure known as the coefficient of variation, and the latter is the kurtosis. FLATMC gives greatest speedup for summations whose terms have low dispersion and low kurtosis. The intuition is that sampling is most efficient when values are concentrated tightly in a few clusters, making it easy to get a representative sample set. This motivates the additional speedup we later gain by stratifying the dataset into low-variance regions through the use of spatial partitioning trees.

Finally, the sample complexity bound indicates whether FLATMC will actually give speedup for any particular problem. For the sake of this argument, let speedup be defined as the total number of summation terms  $n$  divided by the number of sample terms  $m$  evaluated by an invocation of FLATMC. From Theorem 5, we know that with high probability  $m \leq m_{bound}$ , where  $m_{bound}$  is the expression in Equation 26. Overhead aside, the minimum speedup is therefore  $n/m_{bound}$ , so if we desire a speedup of at least  $\tau$ , we need  $m_{bound} \leq n/\tau$ . This is the fundamental characterization of whether a desired level of speedup will be attained. Note that when  $m_{bound}$  is roughly constant with increasing  $n$  (e.g., the i.i.d. case), we can get speedup for any summation, as long as dispersion and kurtosis are finite, once  $n$  is sufficiently large.

### 4.3.3 General Case: Recursive Monte Carlo

We now turn to  $G$ -type nested-summative forms, i.e., Equation 17. The approach we take is to apply the single-stage Monte Carlo algorithm over the terms  $f_i$  as before, but with recursive invocation to obtain approximations for the  $G$ -type arguments  $G_j$  of  $f$ . The procedure, `RECURSIVEMC`, is specified in Algorithm 4.2. Before giving its error bound theorem, we illustrate the operation of `RECURSIVEMC` on the cross-validated kernel regression MSE,  $S_{KR}$ , that was diagrammed in Figure 7.

The algorithm begins as in `FLATMC`, by sampling indices  $i$  from the outermost summation. This is shown in Figure 8 by the points from the outermost summation that feed into the approximation at that level. Each sampled index  $i$  corresponds to a term  $f_i$ , and the sum is approximated by the scaled sample mean  $n\hat{\mu}_f$  as before. Unlike the  $B$ -type forms from `FLATMC`, however, the  $f_i$  of a  $G$ -type form have arguments that are *themselves* nested  $G$ -type forms. In Figure 8, we have  $f_i = 1/n(y_i - G_1(\mathcal{X}_i)/G_2(\mathcal{X}_i))$ , with  $G_1(\mathcal{X}_i) = \sum_{j \neq i} K_h(x_i, x_j)y_j$  and  $G_2(\mathcal{X}_i) = \sum_{j \neq i} K_h(x_i, x_j)$ . These nested  $G$ -type summations must therefore themselves be approximated, which we do by recursive invocation of `RECURSIVEMC` (see `ADDSAMPLESRECURSIVE` in Algorithm 4.2, which maps the recursive evaluation onto each  $G$ -type argument of  $f$  [lines 1(b)-(c)]). This is shown in Figure 8 by the samples feeding into the approximations of  $G_1$  and  $G_2$ . Note that, though  $G_1$  and



**Figure 8:** The `RECURSIVEMC` algorithm performs sampling at each level of nested summation.

---

**Algorithm 4.2** Iterative Monte Carlo approximation for nested summations.

---

RECURSIVEMC

**Input:** type- $G$  summation  $S$ ; constant  $\mathcal{X}_c$ ;  $\epsilon, \delta \in [0, 1]$ ; CLT-sufficient  $m_{min}$

**Output:**  $\hat{S}$  s.t.  $|\hat{S} - S| \leq \epsilon S$  with probability at least  $1 - \delta$

same as FLATMC (Algorithm 4.1), with the exception of line 2(a):

2(a) ADDSAMPLESRECURSIVE( $samples, m_{needed}, S, \mathcal{X}_c$ )

ADDSAMPLESRECURSIVE

**Input:** sample list  $samples$ ; samples to add  $m$ ; type- $G$  summation  $S$ ; constant  $\mathcal{X}_c$

**Output:**  $m$  samples drawn uniformly from the terms of  $S$ , stored in  $samples$

1. **for**  $i = 1$  to  $m$

(a)  $\mathcal{X}_i \leftarrow rand(S.I(\mathcal{X}_c))$

(b) CURRIEDRECMC(\*)  $\leftarrow$  RECURSIVEMC(\*,  $\mathcal{X}_c \circ \mathcal{X}_i, \epsilon, 1, m_{min}$ )

(c)  $GArgVals \leftarrow \mathbf{map}$ (CURRIEDRECMC(\*),  $S.GArgList$ )

(d) APPEND( $S.f(\mathcal{X}_c, \mathcal{X}_i, GArgVals)$ ,  $samples$ )

CALCSTATS

**Input:** sample list  $samples$

**Output:** sample count  $m$ , sample mean  $\hat{\mu}$ , and sample variance  $\hat{\sigma}^2$

same as in Algorithm 4.1

---

$G_2$  are evaluated by recursive invocation of RECURSIVEMC, the recursive calls are made with  $\delta = 1$ . This means there is no error bound on the recursive estimates, but the CLT-sufficiency of  $m_{min}$  ensures that  $\hat{G}_1$  and  $\hat{G}_2$  are normally distributed around the true values, which will turn out to be all we need for the overall error bound.

Once we have approximate values for  $G_1(\mathcal{X}_i)$  and  $G_2(\mathcal{X}_i)$ , we plug them into the expression for  $f_i$  to get an approximate value that serves as a sample term for the outermost summation. The delta method guarantees that the resulting estimates  $\hat{f}_i$  are normally distributed around the true values  $f_i$ , making the overall estimate unbiased and allowing for the same kind of error control as in FLATMC. We now give error control and sample complexity theorems for RECURSIVEMC.

**Theorem 6.** *Given a summation  $S$ ,  $\epsilon \in [0, 1]$ ,  $\delta \in [0, 1]$ , and a value  $m_{min}$  that is CLT-sufficient for both  $S$  and any nested-summative forms contained in  $S$ , with probability at least  $1 - \delta$  the algorithm RECURSIVEMC returns an approximation  $\widehat{S}$  satisfying  $|\widehat{S} - S| \leq \epsilon|S|$ .*

*Proof.* Since RECURSIVEMC is the same as FLATMC except for the recursive sampling procedure called on line 2(a), we piggyback on the proof to Theorem 4. Specifically, Theorem 4 gave the same error guarantee as the present theorem, and depended for its proof on two conditions: 1) the samples come from a distribution with mean  $\mu_f = \frac{1}{n} \sum_i f_i$ , and 2)  $m_{min}$  is CLT-sufficient for the sample mean. We have the second of these as a condition of the present theorem. All we need, then, is to establish that the sampled distribution has mean  $\mu_f$ , i.e., that no bias is introduced by recursive estimation of the nested-summative arguments to  $f$ .

For each sampled index  $i$ , recursive approximation of the arguments to  $f_i$  means that we sample an estimate  $\widehat{f}_i$  rather than the true  $f_i$ . We will show that, under the RECURSIVEMC procedure, the distribution of each sampled  $\widehat{f}_i$  is  $N(f_i, \sigma_i^2)$ , for some  $\sigma_i^2$ . Since we sample the indices  $i$  uniformly, this means the overall distribution being sampled is  $\widetilde{P}_f = \frac{1}{n} \sum_i N(f_i, \sigma_i^2)$ .  $\widetilde{P}_f$  clearly has mean  $\frac{1}{n} \sum_i f_i = \mu_f$ , and this is all that is required to prove the theorem. So all we need to show is that  $\widehat{f}_i \sim N(f_i, \sigma_i^2)$ .

The argument for  $\widehat{f}_i \sim N(f_i, \sigma_i^2)$  is based on chained application of the multivariate delta method. Simply put, the delta method asserts that normality of distribution propagates through a differentiable function.<sup>1</sup> More specifically, if an argument vector  $G$  is drawn from a normal distribution centered on  $\mu_G$ ,  $f(G)$  will be normally distributed around  $f(\mu_G)$ . In our case, for each sampled  $f_i$ , let  $G_j$  be the  $j$ th  $G$ -form argument, and let  $\widehat{G}_j$  be the approximation to  $G_j$  obtained by recursive invocation of RECURSIVEMC. We approximate  $f_i = f(\mathcal{X}_c, \mathcal{X}_i, G_1, G_2, \dots)$  by

---

<sup>1</sup>More precisely, the delta method asserts that *asymptotic* normality propagates through a differentiable function. *Actual* normality is implied as a special case.

$\hat{f}_i = f(\mathcal{X}_c, \mathcal{X}_i, \hat{G}_1, \hat{G}_2, \dots)$ . If we can show that recursive approximation results in  $\hat{G}_j \sim N(G_j, \sigma_{G_j}^2)$ , it follows that the argument vector  $(\mathcal{X}_c, \mathcal{X}_i, \hat{G}_1, \hat{G}_2, \dots)$  is normally distributed around the true value  $(\mathcal{X}_c, \mathcal{X}_i, G_1, G_2, \dots)$ , and therefore that  $\hat{f}_i \sim N(f_i, \sigma_i^2)$ .

We proceed inductively to prove that, under the conditions of the theorem, any invocation of RECURSIVEMC must produce  $\hat{S} \sim N(S, \sigma_S^2)$ . From this it follows that each recursive approximation of  $G_j$  results in  $\hat{G}_j \sim N(G_j, \sigma_{G_j}^2)$ .

The recursion pattern for  $G$ -type summations is a tree, as was illustrated in Figure 8. Leaf nodes represent  $B$ -type summations, while leaves with children are strictly  $G$ -type, meaning that their summand functions require one or more nested-summative arguments. Let the *level* of a leaf node be 0, which we denote by saying the leaf node is  $l_0$ . Let the level of a non-leaf node be 1 plus the maximum level among its children. Thus, in Figure 8, the  $G_1$  and  $G_2$  nodes are  $l_0$ , while the  $S_{KR}$  node is  $l_1$ .

Take  $l_0$  as a base case. The node must be  $B$ -type, and RECURSIVEMC is therefore equivalent to FLATMC for this case. By the CLT-sufficiency of  $m_{min}$  and the same arguments we used in deriving and proving the error guarantee of FLATMC, we know that in this case RECURSIVEMC produces  $\hat{S} \sim N(S, \sigma_S^2)$ .

Now take the general  $l_k$  case. Assume RECURSIVEMC produces  $\hat{S} \sim N(S, \sigma_S^2)$  when invoked on any  $l_q$  node with  $q < k$ . By this inductive hypothesis, all estimates  $\hat{G}_j$  of the arguments  $G_j$  in an  $l_k$  node will satisfy  $\hat{G}_j \sim N(G_j, \sigma_{G_j}^2)$ . By the delta method, we therefore know that the  $\hat{f}_i$  approximations of the  $l_k$  node are sampled from  $N(f_i, \sigma_i^2)$ , and by the CLT-sufficiency of  $m_{min}$  we infer that the estimate  $\hat{S}$  produced by RECURSIVEMC on the  $l_k$  node must be distributed as  $N(S, \sigma_S^2)$ . Since our inductive hypothesis holds at  $l_0$ , we conclude that it holds for arbitrary  $l_k$  and therefore for any invocation of RECURSIVEMC on a  $G$ -type nested-summative form. This establishes the theorem. □

Note that the variance of the induced distribution  $\tilde{P}_f$  works out to

$$\tilde{\sigma}_f^2 = \sigma_f^2 + \frac{1}{n} \sum_{i \in I} \sigma_i^2. \quad (27)$$

In other words, the variance with recursive approximation is the exact-term variance  $\sigma_f^2$  plus the average of the variances  $\sigma_i^2$  of the approximated  $f_i$ . Likewise one could write an expression for the kurtosis  $\tilde{\mu}_{4f}$ . Because RECURSIVEMC is effectively the same as FLATMC but with samples from  $\tilde{P}_f$  instead of  $P_f$ , the direct analogue to Theorem 5 holds straightforwardly for RECURSIVEMC.

**Corollary 7.** *Given a summation  $S$ ,  $\epsilon \in [0, 1]$ ,  $\delta \in [0, 1]$ , and a value  $m_{min}$  that is CLT-sufficient for both  $\hat{\mu}_f$  and  $\hat{\sigma}_f^2$  in  $S$  and for  $\hat{\mu}_f$  in any nested-summative forms contained in  $S$ , with probability at least  $1 - 2\delta$  the algorithm RECURSIVEMC will terminate with sample size  $m \leq O\left(\frac{\tilde{\sigma}_f^2}{\mu_f^2} + \frac{\tilde{\sigma}_f}{|\mu_f|} \sqrt{\frac{\tilde{\mu}_{4f}}{\tilde{\sigma}_f^4} - 1}\right)$ .*

#### 4.3.4 Full Algorithm: Recursive Monte Carlo with Tree-Stratified Sampling

With RECURSIVEMC we have coverage of the entire nested-summative class of computations, and our focus turns to maximizing efficiency. The sample complexity bound in Corollary 7 indicates that the lower the effective dispersion  $\left(\frac{\tilde{\sigma}_f^2}{\mu_f^2}\right)$  and kurtosis  $\left(\frac{\tilde{\mu}_{4f}}{\tilde{\sigma}_f^4}\right)$  of our sampling distribution, the fewer samples we will need to achieve an approximation satisfying the error tolerance. We thus turn to the classical Monte Carlo variance reduction techniques of stratified and correlated sampling. We introduce a new form of stratified sampling driven by spatial partitioning trees derived from the data. This represents the idea of injecting trees into Monte Carlo. We also show how correlated sampling can produce a decrease in variance within our delta method chain. Adding these techniques to RECURSIVEMC makes accurate approximation possible with very small sample sizes. This is true because: 1) stratification greatly reduces the skewness of the sample mean, which, by the Berry-Esséen theorem, allows us to achieve CLT-sufficiency with much smaller values of  $m_{min}$ , and 2)

stratification and correlated sampling reduce variance, which in turn reduces sample complexity (Theorem 5). The combination of these elements thus yields a highly efficient approximation scheme.

#### 4.3.4.1 Stratified Sampling

Stratification is a standard Monte Carlo enhancement designed to reduce the variance of the sample mean. The idea is that the values being sampled are partitioned into subsets (strata) whose contributions are separately estimated and then combined. Strata with higher variance can be sampled more heavily than those with lower variance, thereby making more efficient use of samples than in uniform sampling. Specifically, for a summation  $\sum_{i \in I} f_i = n\mu_f$ , let the index set  $I$  be divided into  $k$  disjoint subsets  $A_1 \dots A_k$ . Let the fraction of index values contained in  $A_i$  be  $p_i = |A_i|/|I|$ . If we take a sample set of total size  $m$ , with  $m_i$  samples coming from the  $i$ th stratum ( $\sum_i m_i = m$ ), the stratified sample mean is:

$$\hat{\mu}_s = \sum_{i=1}^k p_i \hat{\mu}_i, \quad (28)$$

where  $\hat{\mu}_i$  is the mean of the  $m_i$  samples drawn from  $A_i$ . The stratified sample mean is easily shown to be unbiased:

$$E[\hat{\mu}_s] = \sum_{i=1}^k p_i E[\hat{\mu}_i] = \sum_{i=1}^k \frac{n_i}{n} \mu_i = \sum_{i=1}^k \frac{n_i}{n} \frac{1}{n_i} \sum_{j \in A_i} f_j = \frac{1}{n} \sum_{i=1}^n f_i = \mu_f. \quad (29)$$

If we write the fraction of samples allocated to the  $i$ th stratum as  $q_i = m_i/m$ , the variance of  $\hat{\mu}_s$  can be written as:

$$V[\hat{\mu}_s] = \sum_{i=1}^k p_i^2 \frac{\sigma_i^2}{m_i} = \frac{\sigma^2(q)}{m}, \quad (30)$$

where  $\sigma_i^2$  is the variance of the terms within stratum  $i$ , and

$$\sigma^2(q) = \sum_{i=1}^k \frac{p_i^2}{q_i^2} \sigma_i^2. \quad (31)$$

Thus, it is as though we were drawing from an effective distribution with mean  $\mu_f$  and variance  $\sigma^2(q)$ . The stratified variance  $\sigma^2(q)$  is a weighted combination of the per-stratum variances  $\sigma_i^2$ . Clearly, if we can partition the data such that each stratum contains tightly clustered values, the  $\sigma_i^2$  will be small, and so will  $\sigma^2(q)$ . In fact, it is shown in [61] that the effective variance is never greater than the original variance. Low variance is key to the sample efficiency of our Monte Carlo approximation algorithms, as manifest in the sample complexity bound of Theorem 5. It can be shown that the effective variance  $\sigma^2(q)$  is minimized by the optimal allocation  $q^*$  as follows:

$$q_i^* = \frac{p_i \sigma_i}{\sum_{l=1}^k p_l \sigma_l} . \quad (32)$$

In other words, estimation is most efficient when samples are concentrated in large, high-variance strata.

The effective variance  $\sigma^2(q)$  is consistently estimated by:

$$\hat{\sigma}^2(q) = \sum_{i=1}^k \frac{p_i^2}{q_i} \hat{\sigma}_i^2 , \quad (33)$$

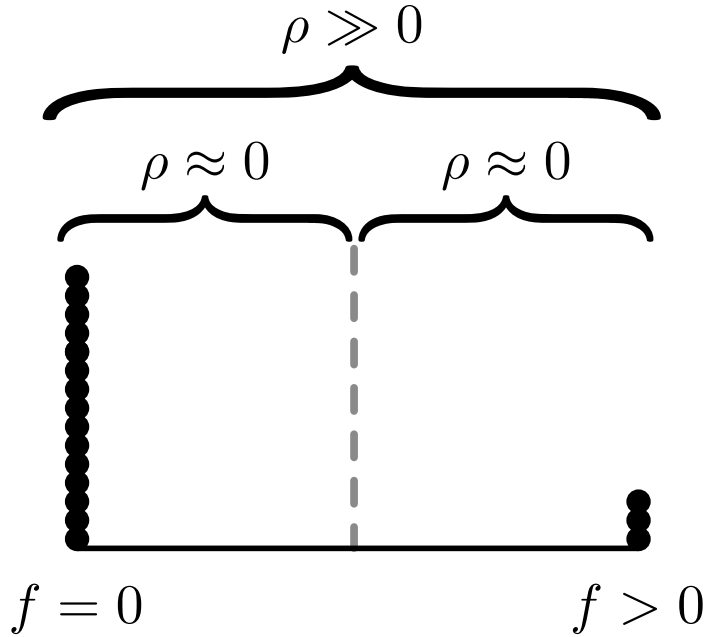
where  $\hat{\sigma}_i^2$  is the sample variance of the samples drawn from stratum  $i$ . It is easily shown that  $\hat{\mu}_s$  obeys the CLT:  $\hat{\mu}_s \rightsquigarrow N(\mu_f, \hat{\sigma}^2(q)/m)$ . We can write the third absolute moment of the stratified sample mean similarly to the variance:

$$\rho[\hat{\mu}_s] = \sum_{i=1}^k \frac{p_i^3}{m_i^2} \rho_i = \frac{\rho(q)}{m^2} , \quad (34)$$

with

$$\rho(q) = \sum_{i=1}^k \frac{p_i^3}{q_i^2} \rho_i . \quad (35)$$

Thus, it is as though we are sampling from an effective distribution with absolute third moment  $\rho(q)$ . For instance, if  $q = p$ , the effective third moment is  $\rho(p) = \sum_i p_i \rho_i$ , a simple weighted average of the per-stratum absolute third moments. This is particularly important in light of the Berry-Esséen theorem, as discussed in Section 4.3.1, which shows that convergence to the CLT is governed by skewness. A good stratification can result in dramatic reduction of skewness, which greatly accelerates CLT



**Figure 9:** Stratification can result in substantially lower per-stratum skewness, which yields low composite skewness.

convergence and yields smaller minimum sample sizes for CLT-sufficiency. As a simple example that occurs often with kernel estimators, consider a sum whose terms are mostly zero except for a few having nearly identical positive values, as illustrated in Figure 9. The base skewness is high, but if stratification is used to separate the zeroes from the positive values, the effective skewness will be reduced to nearly zero.

Thus, in order to both accelerate convergence to CLT-sufficiency and to minimize variance/sample complexity, we would like to employ stratified sampling in our algorithms. Because they obey the CLT, the stratified sample mean and sample variance can be directly substituted into the algorithm RECURSIVEMC. Stratified sampling theory, however, is given only in the abstract — that is, the foregoing results hold in terms of an abstract stratification, but any application of stratified sampling requires the development of an effective partitioning scheme for the domain of interest. In our case, the values being sampled are the  $f_i$ , which are not known *a priori* and cannot be directly stratified. However, since  $f$  is required to be differentiable, it is also continuous and produces similar outputs for similar values of its arguments. We

---

**Algorithm 4.3** Construction of strata based on priority-ordered, fixed-size expansion of a  $kd$ -tree. This  $kd$ -tree variant uses a variance-based expansion priority.

---

CONSTRUCTSTRATA

**Input:** type- $G$  summation  $S$ ; number of strata  $n_{strat}$

**Output:** list of strata  $strata$

1.  $r \leftarrow$  root  $kd$ -tree node containing all data indexed by  $S.I$
  2.  $PQ \leftarrow$  priority queue on  $kd$ -tree nodes s.t. top node has maximum value of: number of data points times sum of per-dimension variances
  3. **while**  $PQ.count < n_{strat}$ 
    - (a)  $topNode = PQ.pop()$
    - (b)  $PQ.insert(topNode.leftChild), PQ.insert(topNode.rightChild)$
  4. **for**  $i = 1$  to  $n_{strat}$ 
    - (a)  $strata[i] = PQ.pop()$
  5. **return** strata
- 

can therefore stratify the argument space, i.e.,  $\mathcal{X}_i$  in Equation 17, to induce a stratification of the  $f_i$ . By sampling from the  $\mathcal{X}_i$  strata and evaluating the resulting  $f_i$ , our samples are drawn from an implicit stratification of the  $f_i$ .

In particular, we want a stratified sampling scheme for kernel estimators over datasets. In this case, the  $\mathcal{X}_i$  are individual data points, and the function  $f$  involves kernel functions that depend on pairwise distances. We therefore choose to stratify the datasets using distance-based spatial partitioning structures. Though any spatial partitioning could be used, in this work we use modified  $kd$ -trees that recursively split the data along the dimension of highest variance. The stratification process is listed in Algorithm 4.3. We expand the  $kd$ -tree with expansion order prioritized by each node’s product of 1) number of points, and 2) sum of per-dimension variances. This heuristic focuses on splitting large, high-variance nodes in order to yield the smallest possible  $\sigma_i^2$ . Expansion continues until the frontier reaches a specified size, at which point the set of frontier nodes constitutes a disjoint stratification of the data.

We inject this tree-based stratification into the RECURSIVEMC algorithm to obtain our final approximation routine, RSMC (Algorithm 4.4). The approximation procedure runs as it did before, except that the samples and sample statistics are computed in stratified fashion. We use Algorithm 4.3 to construct a number of strata  $n_{strat}$  specified by the user, and Algorithm 4.5 to set the allocation  $q$  in each round to approximate the optimal  $q^*$  (Equation 32) based on the estimated stratum variances  $\hat{\sigma}_i^2$ . We now establish that RSMC provides a relative error guarantee similar to those of the previous approximation routines.

**Theorem 8.** *Given a summation  $S$ ,  $\epsilon \in [0, 1]$ ,  $\delta \in [0, 1]$ , and values  $n_{strata}$  and  $m_{min}$  such that  $m_{min}$  is CLT-sufficient for both  $S$  and any nested-summative forms contained in  $S$ , with probability at least  $1 - \delta$  the algorithm RSMC returns an approximation  $\hat{S}$  satisfying  $|\hat{S} - S| \leq \epsilon|S|$ .*

*Proof.* RSMC merely substitutes a stratified sample mean and variance into RECURSIVEMC. These estimators satisfy the same unbiased CLT as the uniform sample mean and variance, and as a condition to the theorem we have the fact that  $m_{min}$  is CLT-sufficient for the stratified estimators. These were the required conditions for the proof of the RECURSIVEMC error bound, so the same proof therefore establishes the RSMC error bound.  $\square$

Finally, the Theorem 5 sample complexity result continues to hold in terms of the *effective* distributional parameters  $\sigma^2(q)$  and  $\mu_4(q)$ . Note that  $q$  here refers to the  $q$  determined by the approximate optimal allocation scheme of Algorithm 4.5.

**Corollary 9.** *Given a summation  $S$ ,  $\epsilon \in [0, 1]$ ,  $\delta \in [0, 1]$ , and a value  $m_{min}$  that is CLT-sufficient for both  $\hat{\mu}_s$  and  $\hat{\sigma}_f^2(q)$  in  $S$  and for  $\hat{\mu}_s$  in any nested-summative forms contained in  $S$ , with probability at least  $1 - 2\delta$  the algorithm RSMC will terminate with sample size  $m \leq O\left(\frac{\hat{\sigma}_f^2(q)}{\mu_f^2} + \frac{\hat{\sigma}_f(q)}{|\mu_f|} \sqrt{\frac{\hat{\mu}_{4f}(q)}{\hat{\sigma}_f^4(q)} - 1}\right)$ .*

---

**Algorithm 4.4** Iterative, recursive, stratified Monte Carlo approximation for nested summations.

---

RSMC

**Input:** type- $G$  summation  $S$ ; constant  $\mathcal{X}_c$ ;  $\epsilon, \delta \in [0, 1]$ ; CLT-sufficient  $m_{min}$ ; number of strata  $n_{strat}$

**Output:**  $\hat{S}$  s.t.  $|\hat{S} - S| \leq \epsilon S$  with probability at least  $1 - \delta$

same as RECURSIVEMC (Algorithm 4.2), with the exception of lines 0 and 2 (a-b):

0.  $strata \leftarrow \text{CONSTRUCTSTRATA}(S, n_{strat})$
2. (a)  $\text{ADDSAMPLESRS}(samples, strata, m_{needed}, S, \mathcal{X}_c)$
2. (b)  $m, \hat{\mu}_f, \hat{\sigma}_f^2 \leftarrow \text{CALCSTATS}(samples, strata)$

ADDSAMPLESRS

**Input:** array of per-stratum sample lists  $samples$ ; stratum list  $strata$ ; samples to add  $m$ ; type- $G$  summation  $S$ ; constant  $\mathcal{X}_c$

**Output:**  $m$  samples approximately optimally allocated, stored in  $samples$

1.  $stratAllocs = \text{OPTALLOC}(samples, strata, m_{needed})$
2. **for**  $s = 1$  to  $strata.count$ 
  - (a) **for**  $i = 1$  to  $stratAllocs[s]$ 
    - i.  $\mathcal{X}_i \leftarrow \text{rand}(S.I(\mathcal{X}_c), strata[s])$
    - ii.  $\text{CURRIEDRSMC}(\ast) \leftarrow \text{RSMC}(\ast, \mathcal{X}_c \circ \mathcal{X}_i, \epsilon, 1, m_{min})$
    - iii.  $GArgVals \leftarrow \text{map}(\text{CURRIEDRSMC}(\ast), S.GArgList)$
    - iv.  $\text{APPEND}(S.f(\mathcal{X}_c, \mathcal{X}_i, GArgVals), samples[s])$

CALCSTATS

**Input:** array of per-stratum sample lists  $samples$ ; stratum list  $strata$

**Output:** sample count  $m$ , stratif. sample mean  $\hat{\mu}$ , and stratif. sample variance  $\hat{\sigma}^2$

1.  $m \leftarrow \text{COUNT}(samples), \hat{\mu} \leftarrow \text{STRATAVG}(samples, strata),$   
 $\hat{\sigma}^2 \leftarrow \text{STRATVAR}(samples, strata)$
  2. **return**  $m, \hat{\mu}, \hat{\sigma}^2$
- 

#### 4.3.4.2 Correlated Sampling

As a final element of efficiency enhancement, we examine more closely the variance  $\tilde{\sigma}_f^2$  induced by recursive sampling, whether stratified or not. Referring back to the proof of Theorem 6, the normality of the estimates  $\hat{f}_i \sim N(f_i, \sigma_i^2)$  comes from the

---

**Algorithm 4.5** Approximate optimal allocation based on sample variances.

---

OPTALLOC

**Input:** array of per-stratum sample lists *samples*; stratum list *strata*; samples to add *m*

**Output:** list of per-stratum allocations *stratAllocs*

1. **if** ISEMPY(*samples*)
    - (a) **return** (*m*, ..., *m*) // initially allocate *m* samples to every stratum
  2. *denom*  $\leftarrow$  0
  3. **for** *i* = 1 to *strata.length*
    - (a) *denom* += *strata*[*i*].*p* \* *strata*[*i*]. $\hat{\sigma}$
  4. **for** *i* = 1 to *strata.length*
    - (a) *stratAllocs*[*i*] = min{1,  $\lceil m * \text{strata}[i].p * \text{strata}[i].\hat{\sigma} / \text{denom} \rceil$ }
  5. **return** *stratAllocs*
- 

delta method and the normality of the argument vector  $\hat{\mathcal{G}}_m = (\mathcal{X}_c, \mathcal{X}_i, \hat{G}_1, \hat{G}_2, \dots)$ , where *m* is the number of samples used to evaluate the recursive summative forms  $\hat{G}_j$ . In fact, the delta method specifies the exact form of the induced variance  $\sigma_i^2$ :

$$\sigma_i^2 = \nabla_f(\mu_{\mathcal{G}}) \Sigma_m \nabla_f^T(\mu_{\mathcal{G}}) , \quad (36)$$

where  $\nabla_f$  is the gradient of the differentiable function *f*,  $\mu_{\mathcal{G}}$  is the mean value of the normally distributed approximate argument vector  $\hat{\mathcal{G}}_m$ , and  $\Sigma_m$  is the covariance of  $\hat{\mathcal{G}}_m$  (see [140]). Because we care about minimizing the variance, we note that  $\sigma_i^2$  depends on the full covariance matrix of the estimated arguments. If the gradient of *f* is such that  $\sigma_i^2$  depends negatively (positively) on a covariance entry  $\Sigma_{m(jk)}$ , we can reduce the variance  $\sigma_i^2$  by inducing positive (negative) covariance between  $G_j$  and  $G_k$ .

In our focus case of kernel estimators,  $G_j$  and  $G_k$  will often be summations of kernel functions over the same datasets. We can therefore induce positive correlation between the estimates  $\hat{G}_j$  and  $\hat{G}_k$  by sharing the sampled indices from one with the other. It should be noted that in some cases the gradient of *f* or the characteristics of a

dataset are such that variance is actually *increased* by inducing correlation. Obviously we would not use correlated sampling in such cases, and it is easy to check empirically whether or not correlation helps. All methods examined in our experiments benefited from correlated sampling, dramatically so in the case of estimating kernel regression MSE scores, due to the quotient form of  $f$  in that case (see Figure 7).

#### 4.4 *Experiments*

Although the final RSMC algorithm has a strong relative error guarantee, the guarantee is contingent on having a sufficiently large number of strata  $n_{strat}$  and initial sample size  $m_{min}$ . Whether these parameters can be set high enough for the guarantee to hold while still yielding a practical, fast approximation algorithm is an empirical question. Our first set of experiments seeks to answer this question by performing RSMC approximations over medium-sized datasets (10–50K points) where exact evaluations are feasible to compute for comparison to our approximations. We use the exact values to test how well the RSMC error bound holds empirically, and measure speedups relative to exact evaluation. Our results show strong satisfaction of the error guarantee while at the same time attaining speedups in the tens and hundreds.

Having established that the RSMC error guarantee can hold with practical sample sizes, we perform a second phase of experimentation to test for scalability. Specifically, we perform RSMC approximations over datasets containing millions and tens of millions of points. Though exact error checking is impossible at this scale, our aim is to gauge the level of speedup in the large-data case. We find that RSMC demonstrates speedups as high as  $10^5$ – $10^6$ .

Our interest is in fast kernel estimators, so we collect measurements over a full bandwidth optimization for two different kernel estimator objectives. Specifically, we optimize over the leave-one-out MSE score for kernel regression ( $S_{KR}$ ), and the leave-one-out ISE score for kernel density estimation ( $S_{KDE}$ ). In both cases we use

the Gaussian kernel. Note that these score functions are an unscalable  $O(n^2)$  to evaluate exactly. Each of the estimators has a single bandwidth parameter to be optimized. To represent a typical optimization procedure, we whiten the data on a per-dimension basis and evaluate the score functions at each bandwidth in the set  $H = \{10^{-4}, 10^{-3}, \dots, 10^2\}$  (see [73] or [80] for similar procedures; the effect is to give each dimension a variable bandwidth proportional to the per-dimensional standard deviation). This generally covers the span from far below to far above the optimal bandwidth, and thus represents the spectrum of bandwidth effects that a full, finer-grained optimization would encounter. All experiments are performed across a diverse array of real datasets in order to test the approximation algorithm under a variety of realistic scenarios. In all cases we use correlated sampling between the recursive summation estimates.

#### 4.4.1 Error Control and Speedup on Medium-Scale Datasets

The objective of this first set of experiments is to validate the guarantee that relative error will be less than or equal to  $\epsilon$  with probability  $1 - \delta$ . This requires the ability to compare to exact answers, and therefore restricts us to medium-sized datasets ranging between 10,000 and 50,000 points. The datasets used in this phase of experiments are described in Table 4. For each dataset and each of the two objective functions ( $S_{KDE}$  and  $S_{KR}$ ), we evaluated the bandwidths in  $H$  both exactly and using RSMC across a range of  $\epsilon$  targets. In all cases we set  $1 - \delta = 0.95$  and determined appropriate settings for  $n_{strat}$  and  $m_{min}$  by trial and error.

We used the values and runtimes of the exact computations to determine the relative errors and speedups of the RSMC approximations. Figure 10 summarizes the error performance by listing, for each dataset and each objective function, the fraction of invocations that returned values at or below the error target. As a baseline, we employed the subsample-and-run-exact technique on subsamples of size 50. The

**Table 4:** Descriptions of the six datasets used for medium-scale experiments, ranging from 17,605 to 50,000 in size and from 3 to 9 in dimension.

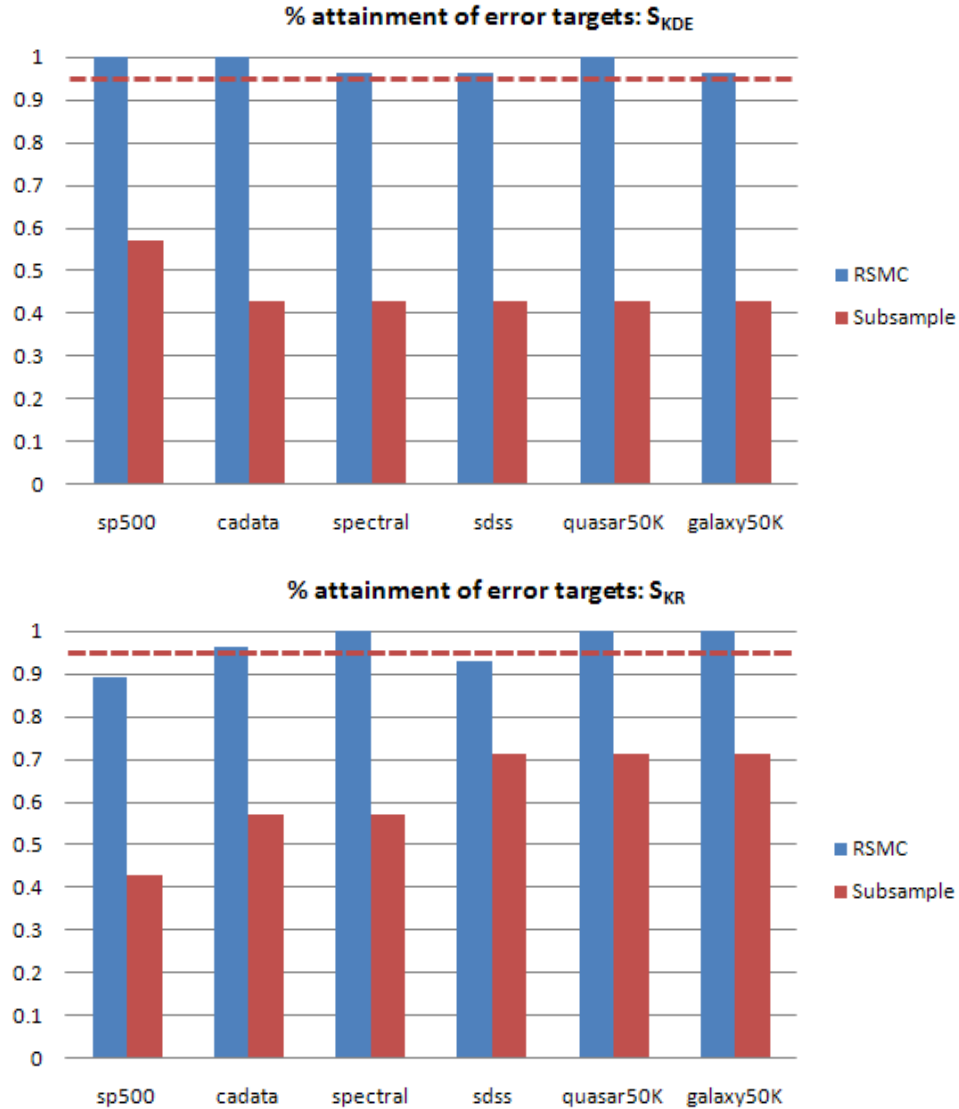
name	size	dimension	notes
sp500	17,605	6	stock time series
cadata	20,640	9	census data
spectral	46,420	5	red-shift spectral measurements
sdss	50,000	3	Sloan Digital Sky Survey objects
quasar50K	50,000	4	quasar identification dataset
galaxy50K	50,000	3	output of a galaxy simulation

fraction of subsampling invocations that meet the highest  $\epsilon$  targets from the RSMC invocations are shown for comparison.<sup>2</sup> The salient things to note in these results are that RSMC is consistently able to meet its error guarantee, and that this is much better performance than the naive subsample-and-run-exact baseline. The difference would be even more stark if we included only near-optimal bandwidths, in which case the subsampling fractions would drop nearly to zero. Also, though not apparent from Figure 10, when RSMC missed its error target it was always by a relatively small amount, whereas when subsampling missed the target, it was usually by a large margin. Altogether, RSMC has thus demonstrated very reliable error control.

The speedup of the RSMC invocations over a range of  $\epsilon$  values are displayed in Figures 12–23 at the end of the chapter. As expected, in all cases the speedup factor increases substantially with  $\epsilon$ , ranging from the tens to the hundreds across the various datasets and error levels. While this is decent speedup performance, representing reductions from hours to seconds or minutes of computation, the true strength of RSMC lies in its dependence on dataset complexity rather than dataset size, and this can only be seen at scale.

---

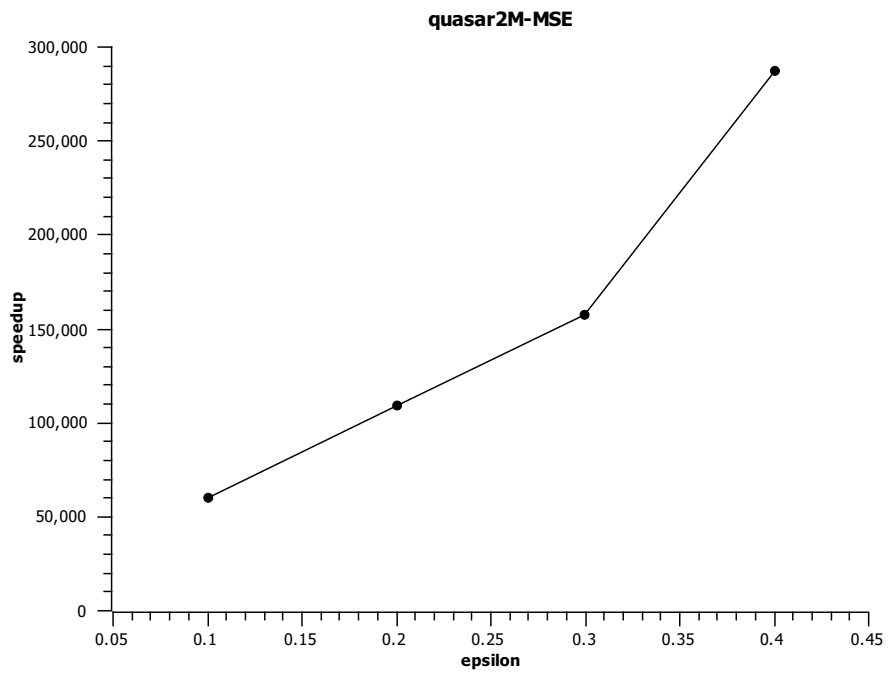
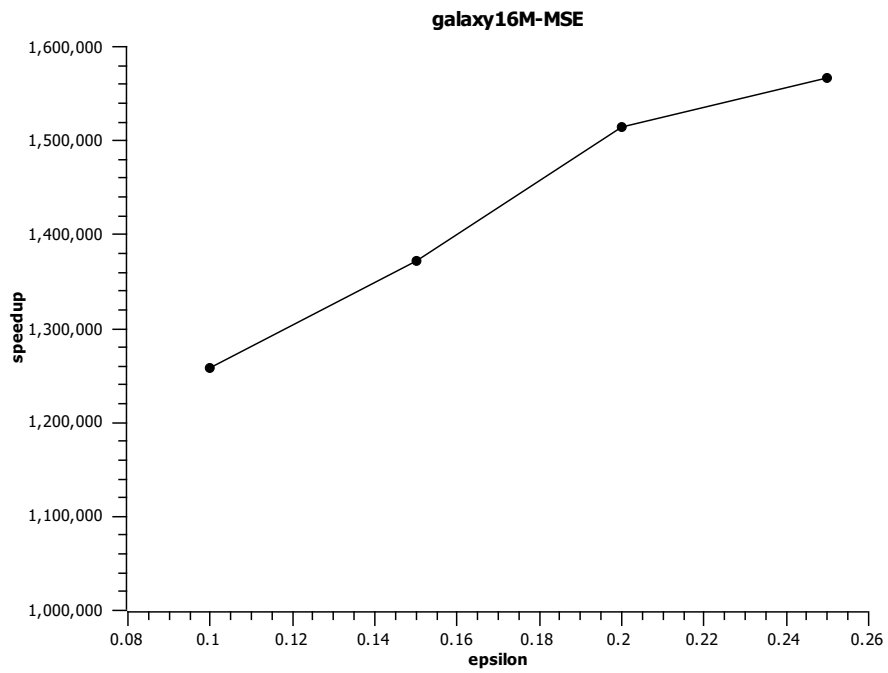
<sup>2</sup>Though the fractions for subsampling may in some cases seem surprisingly high, this is because  $H$  includes bandwidths far from the optimal, which are so large or small as to make most kernel evaluations equal, in which case sampling any few of them gives a reasonable approximation.



**Figure 10:** Percentages of runs achieving their  $\epsilon$  relative error target for RSMC and simple subsampling on  $S_{KDE}$  (upper) and  $S_{KR}$  (lower). The target percentage is  $1 - \delta = 0.95$ , as represented by the dashed line.

#### 4.4.2 Scalability on Large Datasets

Having given empirical validation of the error guarantees, we now turn to computational performance in the large data case. The setup is identical to that of the previous section, except that the datasets are now too large for exact evaluation to be practical. The particular datasets we use here, quasar2M and galaxy16M, are supersets of the quasar50K and galaxy50K datasets, and contain 2 million and 16



**Figure 11:** Speedup vs. relative error tolerance for  $S_{KR}$  on the galaxy16M (upper) and quasar2M (lower) datasets.

million points, respectively. We extrapolate exact runtimes based on their  $O(n^2)$  growth order and the known runtimes from quasar50K and galaxy50K. Again, all times measured are for the entire optimization over  $H$ . We plot the speedup of RSMC approximation over extrapolated exact runtimes in Figure 11. The results are significant, with speedups on the order of  $10^4$  to  $10^5$  for quasar2M, and on the order of  $10^6$  for galaxy16M. In absolute terms, RSMC approximation reduced the runtimes from nearly 3 years to half an hour for quasar2M, and from nearly 50 years to 20 minutes for galaxy16M. These speedups represent several-order-of-magnitude improvement over the prior state of the art, and open the way to applications with massive data that have previously been impossible.

## 4.5 Conclusion

We have presented a recursive stratified Monte Carlo method, RSMC, for efficiently approximating a broad class of nested-summative computational forms. Summations of this type form the key  $O(n^2)$ + computational bottlenecks in kernel estimators and elsewhere in machine learning. RSMC injects trees into Monte Carlo to perform highly efficient stratified sampling. It also automatically controls relative error to fall within user-specified tolerances with high probability, given a sufficient initial sample size. We have shown a sample complexity bound for RSMC estimation that depends only on statistical features of the input data, and not on dataset size directly. This is particularly attractive for the large-data case, which is our focus. Empirically, our results validate the theoretical error guarantees on real datasets, and the algorithm has produced speedup factors as high as  $10^5$ – $10^6$  on datasets with as many as 16 million points. This is several orders of magnitude faster than the previous state of the art, and should enable new applications and even the derivation of new methods that might have otherwise not been considered due to impracticality.

There remain significant directions in which the method presented here could be

extended. Unconditional small-sample error control and automatic determination of stratification granularity are chief among these, and would eliminate the tweak parameters of the algorithm. Nonetheless, RSMC has shown impressive promise and could become the go-to method for training kernel estimators on massive data.

Having brought the Multi-Tree Monte Carlo methodology to bear on kernel estimators and the general nested-summative class, we now turn to a very different computational type, the linear algebraic, in the form of the SVD. Though a vastly different type of computation, we will find that similar principles can be brought to bear, resulting in similar computational acceleration. This will in turn push us toward the general form of the Multi-Tree Monte Carlo methodology, providing a glimpse of how broadly it may be applied.

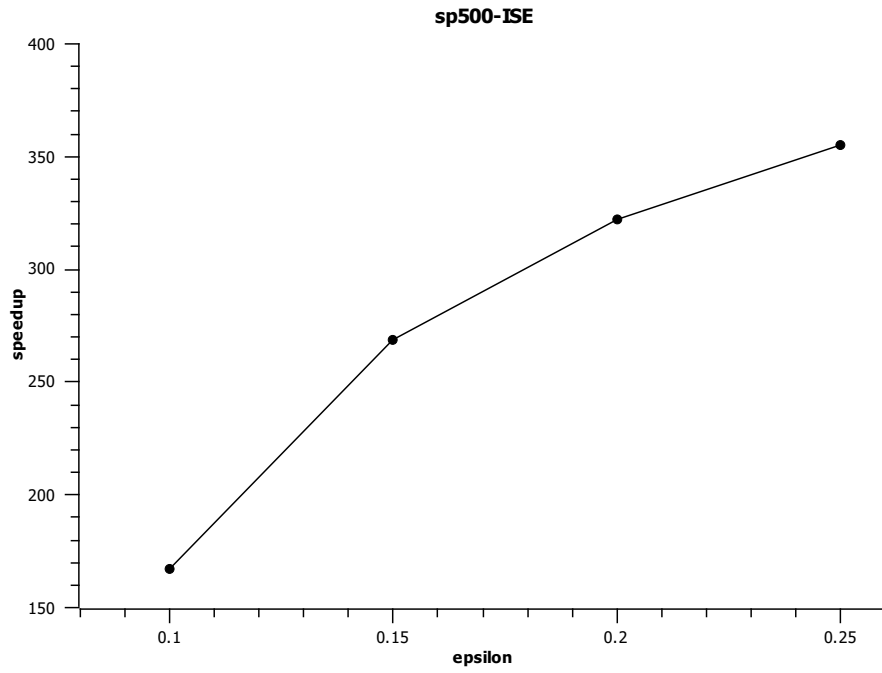


Figure 12: Speedup vs. relative error tolerance for  $S_{KDE}$  on the sp500 dataset.

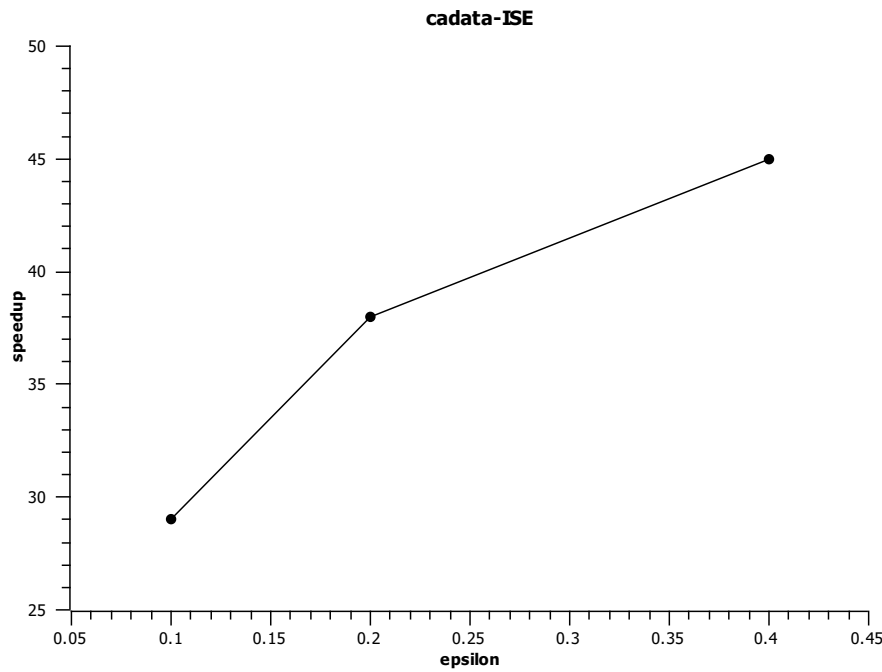
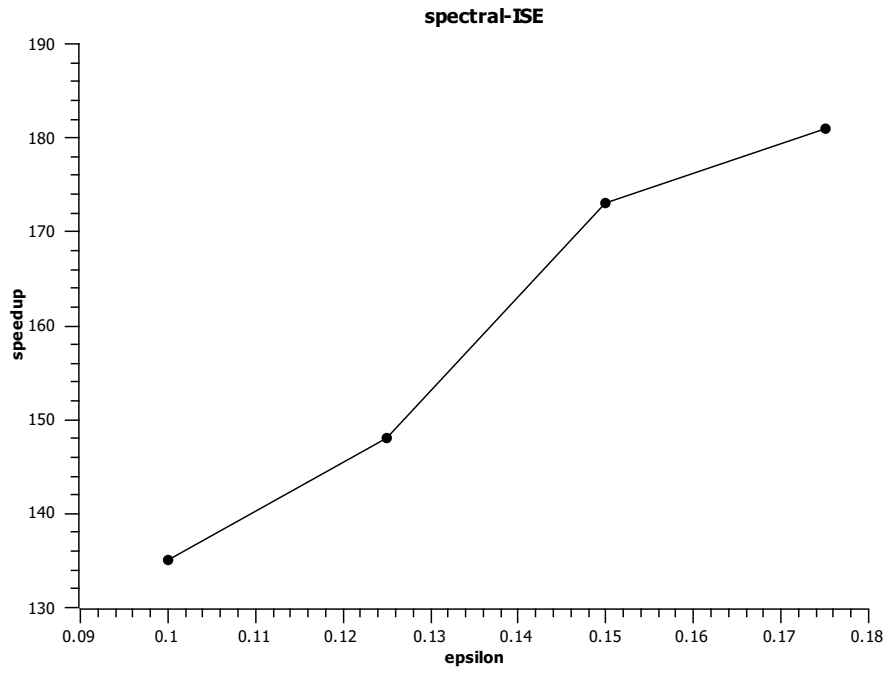
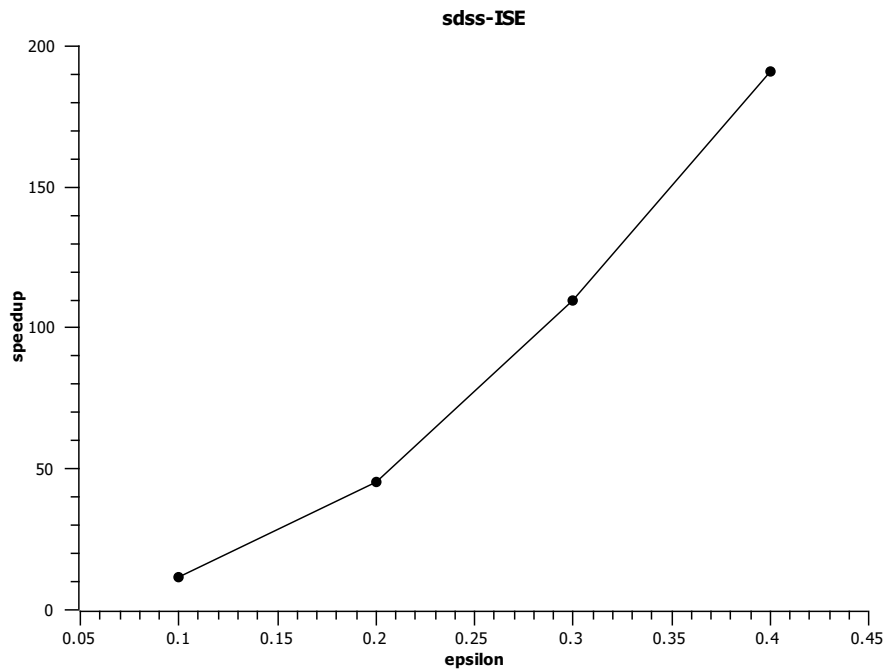


Figure 13: Speedup vs. relative error tolerance for  $S_{KDE}$  on the cadata dataset.



**Figure 14:** Speedup vs. relative error tolerance for  $S_{KDE}$  on the spectral dataset.



**Figure 15:** Speedup vs. relative error tolerance for  $S_{KDE}$  on the sdss dataset.

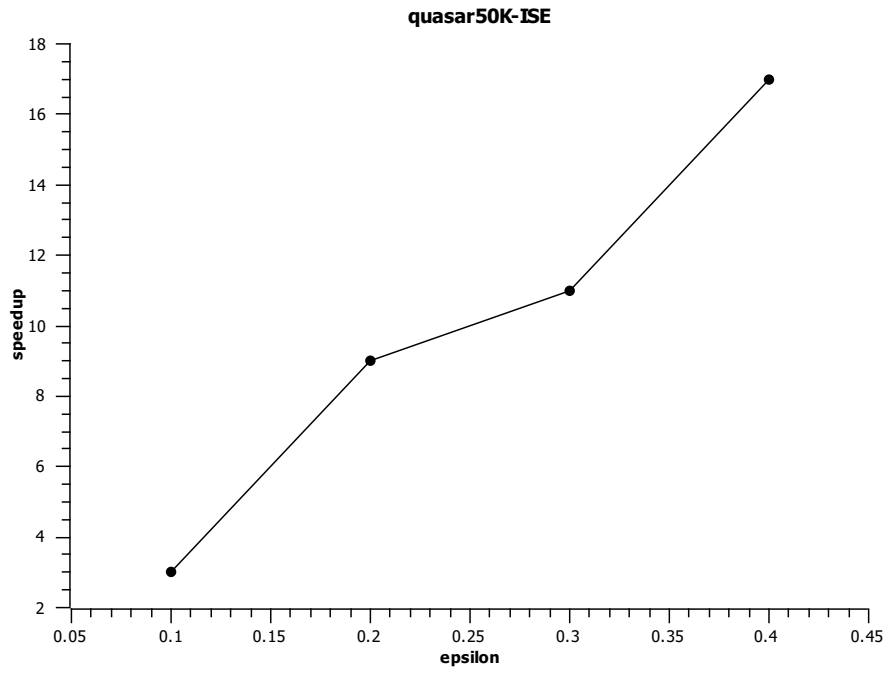


Figure 16: Speedup vs. relative error tolerance for  $S_{KDE}$  on the quasar50K dataset.

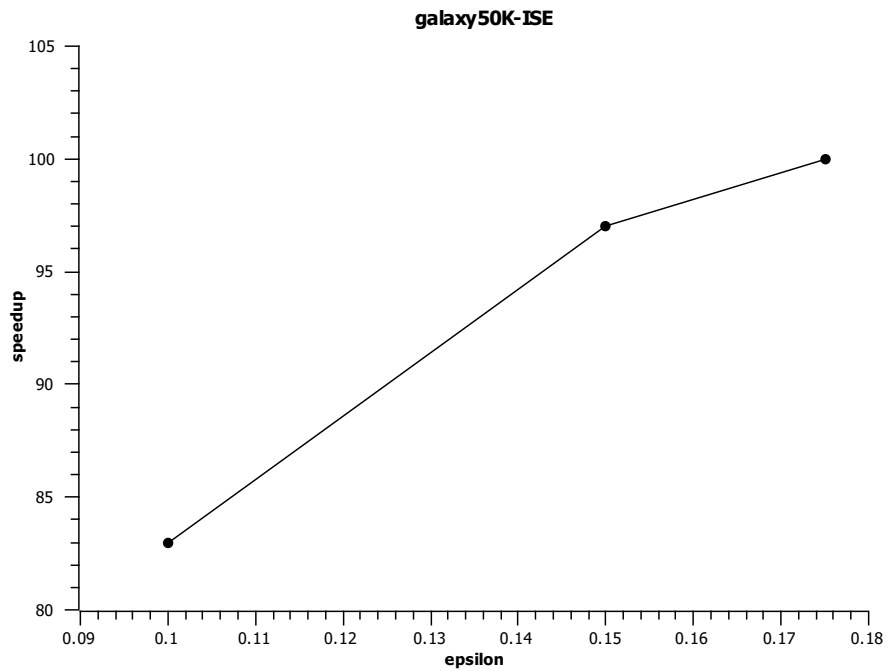
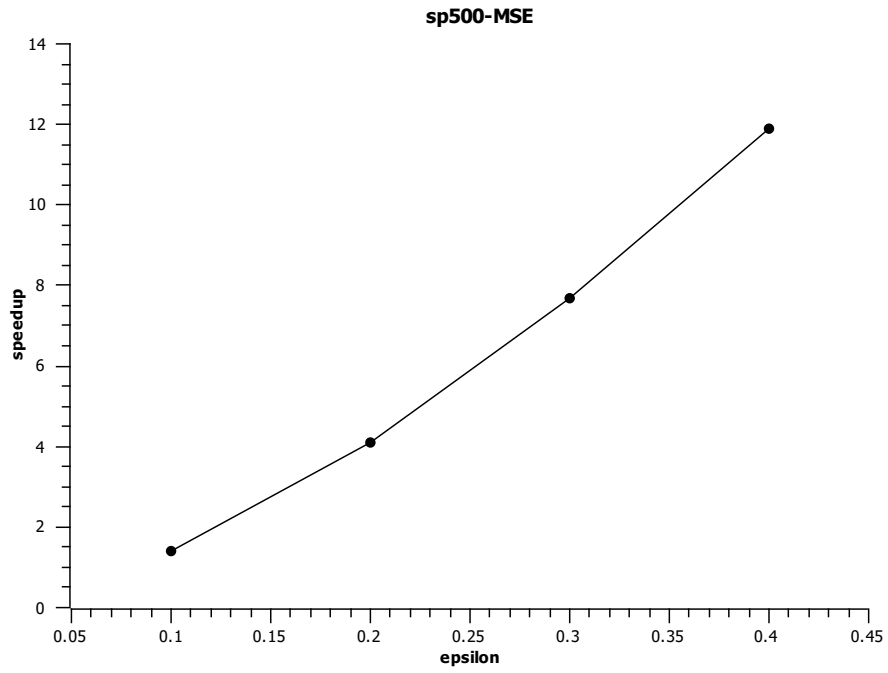
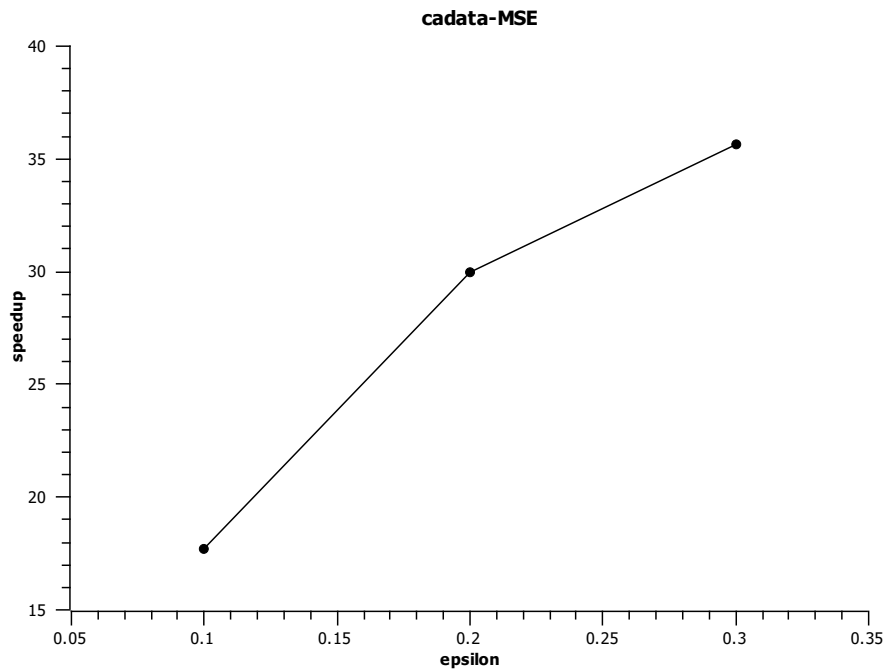


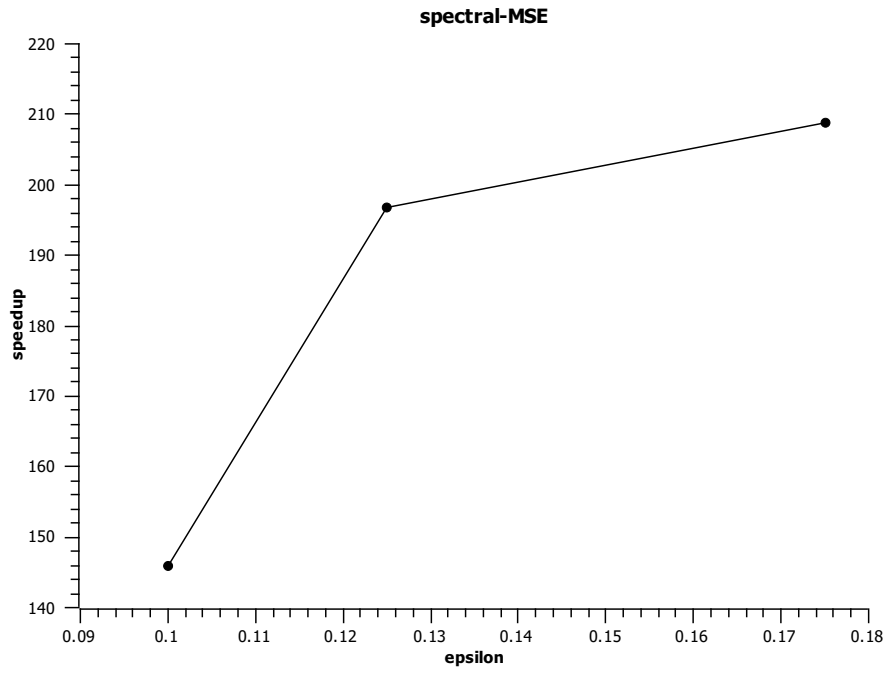
Figure 17: Speedup vs. relative error tolerance for  $S_{KDE}$  on the galaxy50K dataset.



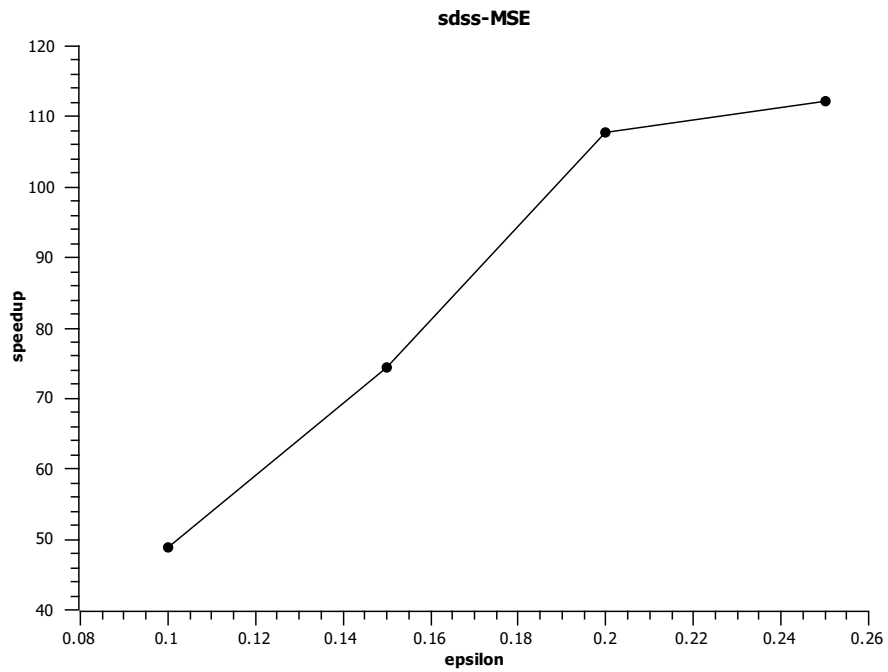
**Figure 18:** Speedup vs. relative error tolerance for  $S_{KR}$  on the sp500 dataset.



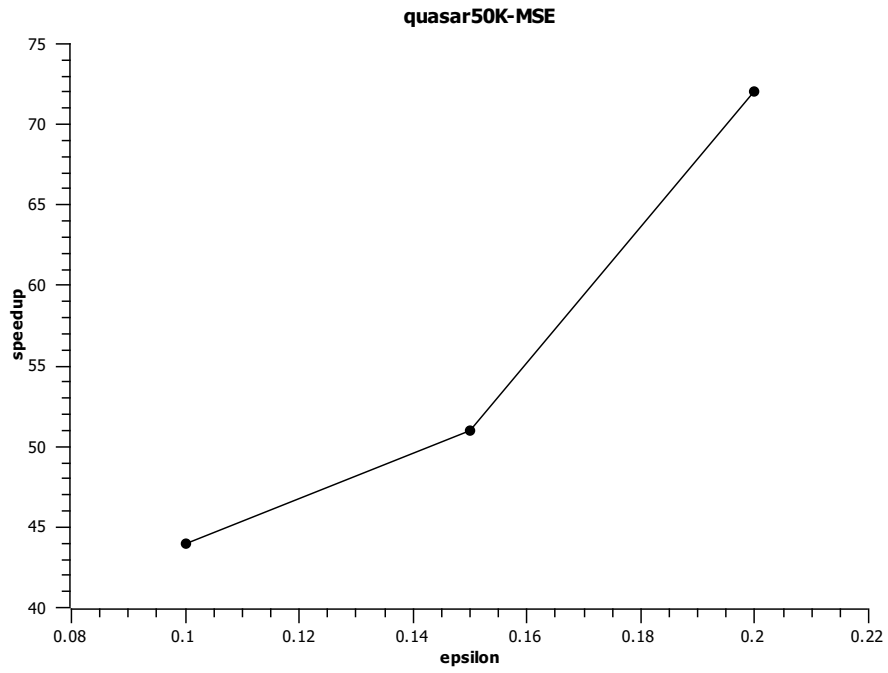
**Figure 19:** Speedup vs. relative error tolerance for  $S_{KR}$  on the cadata dataset.



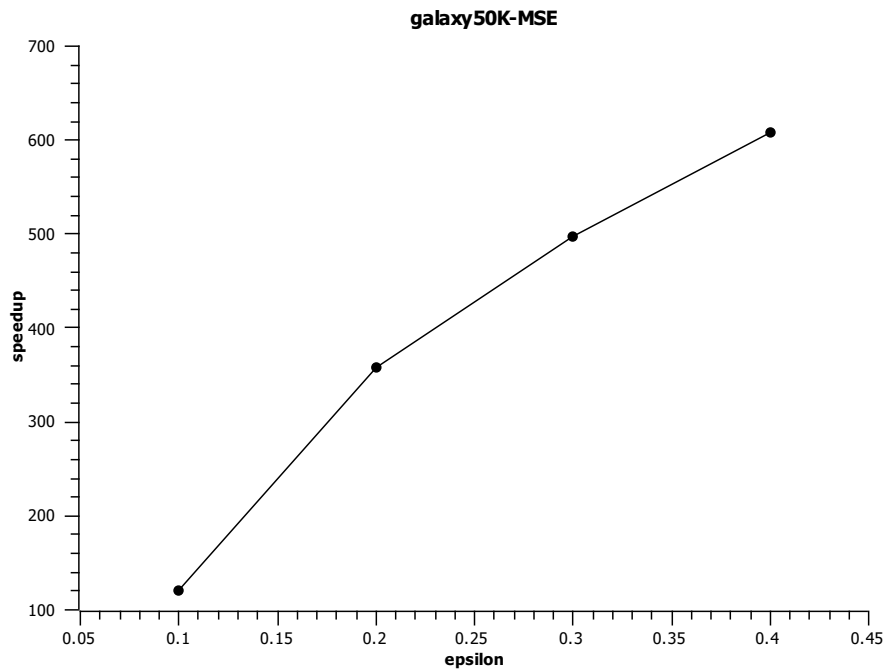
**Figure 20:** Speedup vs. relative error tolerance for  $S_{KR}$  on the spectral dataset.



**Figure 21:** Speedup vs. relative error tolerance for  $S_{KR}$  on the sdss dataset.



**Figure 22:** Speedup vs. relative error tolerance for  $S_{KR}$  on the quasar50K dataset.



**Figure 23:** Speedup vs. relative error tolerance for  $S_{KR}$  on the galaxy50K dataset.

## CHAPTER V

### RECIPROCAL TREES AND MONTE CARLO: FAST SINGULAR VALUE DECOMPOSITION

Having demonstrated the effectiveness of Multi-Tree Monte Carlo in accelerating machine learning methods with summative computational forms, we now turn to a very different type of computation: the Singular Value Decomposition (SVD). The SVD is a key linear algebraic operation at the heart of many methods both in machine learning and data mining, as well as many other fields. Its computational cost, however, makes it unscalable and impractical for the massive-sized datasets becoming common in applications. While the SVD is of a fundamentally different computational type than the generalized summations previously addressed, we show that the Multi-Tree Monte Carlo approach from Chapters 3–4 can be adapted to produce a fast, error-controlled approximation. Specifically, we present a new method, QUIC-SVD, for fast approximation of the whole-matrix SVD with automatic sample size minimization and empirical relative error control [82]. Though we draw upon recent low-rank matrix approximation ideas from the theory and algorithms community, previous approaches have not addressed the whole-matrix SVD nor benefited from the efficiency of automatic, empirically-driven sample sizing. Experimentally, we show that QUIC-SVD attains speedups on the order of  $10^3$ – $10^5$  over exact SVD on matrices containing billions of entries. Such speedups represent computation times being reduced from years to hours and from days to minutes. Thus, QUIC-SVD shows the kind of scalability that should meet the needs of a wide array of methods and applications, especially those dealing with large-scale data, repeated invocations, time-sensitivity, etc. The adaptation of Multi-Tree Monte Carlo to the linear-algebraic SVD will also

lead us to a generalized formulation of the Multi-Tree Monte Carlo methodological framework, which in turn suggests how we might go about developing fast approximations for methods beyond those considered in this thesis.

## ***5.1 Introduction***

The Singular Value Decomposition (SVD) is a fundamental linear algebraic operation whose abundant useful properties have placed it at the computational center of many methods in data mining and machine learning. Among such methods, principal components analysis (PCA), along with its kernel and other nonlinear variants [128, 122] are perhaps the most well known, but all methods involving the eigendecomposition of a symmetric, positive semidefinite matrix can be computed by the SVD. Linear systems, least-squares problems, and matrix inversions can be solved by the SVD, leading to its use in methods for manifold and metric learning, clustering, natural language processing/search, collaborative filtering, bioinformatics and more [54, 57, 34, 107, 27, 10, 62, 43, 18, 51, 50]. Indeed, though it is sometimes overkill, the SVD can be used to solve essentially any problem in numerical linear algebra.

Notwithstanding the utility of the SVD, it is critically bottlenecked by a computational complexity that renders it intractable on massive datasets. Yet massive datasets are rapidly becoming the norm in applications, many of which require real-time responsiveness or would use the SVD more liberally if it were not so slow to compute. We present a new method, QUIC-SVD, that provides a fast approximation of the whole-matrix SVD with automatic sample size minimization and empirical relative error control. Error tolerance is specified by the user, and QUIC-SVD automatically produces an approximation within the user-specified tolerance. The algorithm is based on a new type of data partitioning tree, the cosine tree, that shows excellent ability to home in on the data subspace needed for good SVD approximation.

Our approach draws on a recent vein of work concentrated in the theory and algorithms community, in which fixed low-rank matrix approximations are obtained by sampling. In contrast to such methods, we are interested the actual SVD rather than a matrix approximation, and we address the whole-matrix case (i.e., with quality approximately as good as full-rank) rather than a fixed low-rank approximation (which can be of much lower than full-rank quality). These distinctions are critical for many, if not most, applications. We show that QUIC-SVD runs in time  $O(kmn)$  on an  $m \times n$  matrix, where  $k$  is the rank required to get within the user-specified error tolerance ( $k$  is determined automatically by the algorithm; its minimum value depends on spectral structure of the matrix). We demonstrate empirically that sampling based on cosine trees is dominantly more efficient than any previously proposed sampling scheme, and that QUIC-SVD attains speedups on the order of  $10^3$ – $10^5$  on dataset matrices containing billions of entries, while correctly keeping error to within user-specified tolerances. Based on these results, we conclude that QUIC-SVD should be able to help address the scale of modern problems and datasets, with the potential to benefit a wide array of methods and applications.

## 5.2 Notation and SVD Background

For  $A \in \mathbb{R}^{m \times n}$ , we write  $A_{(i)}$  for the  $i$ th row of  $A$  and  $A^{(j)}$  for the  $j$ th column. We use  $\mathbb{O}^{m \times n}$  to represent the subset of  $\mathbb{R}^{m \times n}$  whose columns are orthonormal. Since the columns of  $V \in \mathbb{O}^{m \times n}$  are an orthonormal basis, we sometimes use expressions such as “the subspace  $V$ ” to refer to the subspace *spanned* by the columns of  $V$ .

Throughout this chapter we assume  $m \geq n$ , such that our sampling-based method gives bigger speedup by sampling rows than columns. This is no loss of generality, since whenever  $m < n$  we can perform SVD on the transpose, then swap  $U$  and  $V$  to get the SVD of the original matrix. Alternatively, row-sampling-based methods have analogous column-sampling versions that can be used in place of transposition;

however, we leave these implicit and develop only the row-sampling versions.

The following theorem defines the singular value decomposition:

**Theorem 10.** *Let  $A$  be an  $m \times n$  real matrix of rank  $\rho$ . Then there exists a factorization of the form*

$$A = U\Sigma V^T, \quad (37)$$

where  $U \in \mathbb{O}^{m \times \rho}$  and  $V \in \mathbb{O}^{n \times \rho}$  are orthogonal matrices of sizes  $m \times \rho$  and  $n \times \rho$ , respectively, and  $\Sigma$  is diagonal with entries  $\sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_\rho > 0$ .

Equivalently, we can write the SVD as a weighted sum of rank-one outer products:

$$A = \sum_{i=1}^{\rho} \sigma_i u_i v_i^T, \quad (38)$$

where  $u_i$  and  $v_i$  represent the  $i$ th columns of  $U$  and  $V$ . The columns  $u_i$  and  $v_i$  are referred to as the left and right singular vectors, while the weights  $\sigma_i$  are the singular values. With  $m \geq n$ , the computational cost of the SVD is  $O(mn^2)$ . The following relations also hold:  $Av_i = \sigma_i u_i$ ,  $A^T u_i = \sigma_i v_i$ ,  $\|Av_i\|_F = \|A^T u_i\|_F = \sigma_i$ . If  $A$  is (symmetric) positive definite, it follows that: 1)  $U = V$ , 2) the SVD is the same as the eigendecomposition, and 3) the singular vectors/values are equal to the eigenvectors/values. In particular,  $AA^T = U\Sigma^2 U^T$  and  $A^T A = V\Sigma^2 V^T$ , so the right singular vectors are the eigenvectors of  $AA^T$ , the left singular vectors are the eigenvectors of  $A^T A$ , and the squared singular values are the eigenvalues of both  $AA^T$  and  $A^T A$ . Thus, the SVD can be used to solve eigenproblems such as PCA.

Often we are interested in lower-rank approximations to the full-rank matrix  $A$ . It can be shown (Eckart-Young theorem) that the optimal  $k$ -rank approximation, in the sense of minimizing  $\|A - \tilde{A}\|_F^2$ , is the  $k$ -rank truncation of the SVD:

$$A_k = \sum_{i=1}^k \sigma_i u_i v_i^T = U_k \Sigma_k V_k, \quad (39)$$

where  $U_k$  contains the top  $k$  left singular vectors ( $m \times k$ ),  $V_k$  contains the top  $k$  right singular vectors ( $n \times k$ ), and  $\Sigma$  contains the top  $k$  singular values ( $k \times k$ ).  $A_k$

represents the projection of the rows/columns of  $A$  onto the subspace spanned by the top  $k$  right/left singular vectors, i.e.,  $A_k = AV_kV_k^T = U_kU_k^T A$ . Thus, the optimality of  $A_k$  implies that the columns of  $V_k$  and  $U_k$  span the  $k$ -rank row and column subspaces in which  $A$ 's projection has minimum squared error. Equivalently, the subspaces spanned by  $V_k$  and  $U_k$  maximize the Frobenius norm of  $A$ 's projection onto them:

$$V_k = \operatorname{argmax}_{\tilde{V} \in \mathbb{O}^{n \times k}} \|A\tilde{V}\tilde{V}^T\|_F^2 \quad (40)$$

$$U_k = \operatorname{argmax}_{\tilde{U} \in \mathbb{O}^{m \times k}} \|\tilde{U}\tilde{U}^T A\|_F^2. \quad (41)$$

This formulation is suggestive of an SVD approximation scheme in which we seek first to find a lower-dimensional subspace that captures enough of  $A$  to attain a desired error tolerance, then perform the SVD of  $A$ 's projection onto the subspace. An important procedure we will require is the extraction of the best approximate SVD to  $A$  from within a subspace  $\widehat{V}$ . Algorithm 5.1 describes this procedure; portions of this idea appeared in [51] and [31], but without enumeration of its properties. We state some of the key properties as a lemma.

**Lemma 4.** *Given a target matrix  $A$  and a row subspace basis contained in the columns of  $\widehat{V}$ , EXTRACTSVD has the following properties:*

1. *Returns a full SVD, meaning  $U$  and  $V$  with orthonormal columns, and  $\Sigma$  diagonal.*
2.  *$U\Sigma V^T = A\widehat{V}\widehat{V}^T$ , i.e., the extracted SVD reconstructs exactly to the projection of  $A$ 's rows onto the subspace spanned by  $\widehat{V}$ .*
3.  *$U\Sigma V^T$  has optimal squared-error reconstruction of  $A$  among all SVDs whose rows are restricted to the span of  $\widehat{V}$ .*

*Proof.* Properties 1 & 2.  $A\widehat{V}$  is the projection of  $A$  onto  $\widehat{V}$ , expressed in the coordinates of  $\widehat{V}$ 's basis. By computing the SVD of  $(A\widehat{V})^T A\widehat{V} = U'\Sigma'V'^T$ , we obtain the

---

**Algorithm 5.1** Optimal approximate SVD within a row subspace  $\widehat{V}$ .

---

EXTRACTSVD

**Input:** target matrix  $A \in \mathbb{R}^{m \times n}$ , subspace basis  $\widehat{V} \in \mathbb{O}^{n \times k}$ **Output:**  $U\Sigma V^T$ , the SVD of the best approximation to  $A$  within the subspace spanned by  $\widehat{V}$ 's columns.

1. Compute  $A\widehat{V}$ , then  $(A\widehat{V})^T A\widehat{V}$  and its SVD:  $U'\Sigma'V'^T$ .
  2. Let  $V = \widehat{V}V'$ ,  $\Sigma = (\Sigma')^{1/2}$ , and  $U = (A\widehat{V})V'\Sigma^{-1}$ .
  3. Return  $U, \Sigma, V$ .
- 

SVD row basis and squared singular values of  $A\widehat{V}$ . Left-multiplying  $V'$  by  $\widehat{V}$  rotates the SVD row basis back to the original coordinate system, and taking the square root of  $\Sigma'$  gives us the correct singular values. With  $V$  and  $\Sigma$  in hand, we compute  $U = AV\Sigma^{-1}$ . Because  $V = \widehat{V}V'$  and we have already computed  $A\widehat{V}$ , we get this by  $U = (A\widehat{V})V'\Sigma^{-1}$  (note that we avoid recomputation of  $A\widehat{V}$ ). Thus, EXTRACTSVD returns the SVD of  $A$ 's projection onto  $\widehat{V}$ , and it is a true SVD.

*Property 3.* It is a basic geometrical fact that the optimal squared-error approximation to  $A$  within  $\widehat{V}$  is  $A\widehat{V}\widehat{V}^T$ . By property 2, EXTRACTSVD returns the SVD of  $A\widehat{V}\widehat{V}^T$ . Thus, the SVD it returns is optimal in squared-error reconstruction of  $A$ , given the restriction to the subspace of  $\widehat{V}$ .  $\square$

Note that the runtime of EXTRACTSVD is  $O(kmn + k^2(m + n))$  for the various matrix products, plus  $O(k^3)$  for the SVD of  $(A\widehat{V})^T A\widehat{V}$ . Since  $k \leq n \leq m$ , this simplifies to  $O(kmn)$ , which is a reasonable linear scaling in the number of matrix terms  $mn$ . In contrast, the  $O(mn^2)$  cost of computing the exact SVD represents quadratic scaling in terms of one of the matrix dimensions (for a dataset, this is quadratic in either the number of points or number of dimensions). Though feasible for small-to-medium-sized datasets or offline processing, many applications involve real-time learning and/or data matrices with dimensions ranging from tens of thousands to

millions and beyond. Quadratic scaling is simply too slow. We need a fast approximation, but the key is to produce an SVD that is not just fast, but controllably accurate so that it can be relied on as a component in sophisticated, higher-level algorithms. If we could efficiently find a suitable subspace for such an SVD, the SVD would be computable in  $O(kmn)$  time by EXTRACTSVD. This is the launch point for QUIC-SVD.

### 5.3 Previous Work

Our approach draws upon ideas from a recent vein of work in the theory and algorithms community, in which subsampling is used to get a low-rank approximation to an input matrix. While low-rank matrix approximation (LRMA) is different from approximation of the whole-matrix SVD, it is closely related and similar techniques can serve both problems. We summarize the main results from the LRMA work, then describe the distinctions between the problem of LRMA and our problem of whole-matrix SVD approximation, and between our methods and those used in LRMA.

#### 5.3.1 Low-Rank Approximation with Additive Error Control

The idea of low-rank matrix approximation is that the user specifies a fixed low rank  $k$ , and the algorithm tries to output a matrix close to the optimal  $k$ -rank approximation of the input matrix  $A$ . As noted earlier, the optimal  $k$ -rank approximation is the  $k$ -truncated SVD  $A_k = \sum_{i=1}^k \sigma_i u_i v_i^T$ , which shows a connection between LRMA and the SVD. The work of Frieze, Vempala, and Kannan [53, 52] originated the insight that LRMA could be solved via sampling methods. They use a subsample of rows and columns to generate a  $k$ -rank subspace onto which the input matrix can be projected with boundable error. Sampling is performed with respect to the length-squared distribution, i.e., the probability of sampling a row/column is proportional to its squared length:  $p_i \propto \|A_{(i)}\|_F^2 / \|A\|_F^2$ . This is essentially an importance-sampling procedure, and is shown to minimize variance among fixed-probability sampling schemes.

As a quality-control guarantee, the Frieze et al. work shows that if the number of sampled rows/columns is  $O(\max\{\frac{k^4}{\epsilon^2}, \frac{k^2}{\epsilon^4}\})$ , the low-rank approximation  $\widehat{A}$  will satisfy the following additive error bound:

$$\|A - \widehat{A}\|_F^2 \leq \|A - A_k\|_F^2 + \epsilon \|A\|_F^2. \quad (42)$$

Since  $A_k$  is the optimal  $k$ -rank approximation,  $\|A - A_k\|_F^2$  is an absolute lower bound on  $\|A - \widehat{A}\|_F^2$ .  $\|A\|_F$ , however, can be arbitrarily large relative to  $\|A - A_k\|_F$ , so this additive bound is of questionable utility.  $\widehat{A}$  is returned implicitly in the form of a  $k$ -orthonormal basis  $\widehat{V} = [v_1 \dots v_k]$ , which is an approximation to the optimal  $V_k$ ;  $\widehat{A}$  would be computed by projecting  $A$  onto this subspace:  $\widehat{A} = A\widehat{V}\widehat{V}^T$ . The algorithm is said to run in “constant”  $\text{poly}(k, \frac{1}{\epsilon})$  time; however, this excludes the  $O(mn)$  cost of computing the length-squared distribution, as well as the  $O(kmn)$  cost that would be required to compute  $\widehat{A}$  from its “description”  $\widehat{V}$ . One additional item of note is that the row/column sampling scheme is numerically unstable in practice, requiring an extra filtering step in the algorithm.

Most of the follow-on work has consisted of improvements to the sample complexity while retaining the additive error bound. Drineas et al. [40, 39] derive a  $O(k/\epsilon^2)$  sample complexity for sampling columns only, and simplify the row/column sample complexity to  $O(k^2/\epsilon^4)$ . They also give a spectral-norm error bound analogous to Equation 42 at sample complexities of  $O(1/\epsilon^2)$  (column-only) and  $O(1/\epsilon^4)$  (row/column). Runtime in the column-sampling case acquires an additional linear factor of  $\max\{m, n\}$ . Although Drineas et al. refer to their algorithms as `LIN-EAR-TIMESVD` and `CONSTANT-TIMESVD`, they do not actually compute an SVD, but an approximation to the singular values and left singular vectors only (i.e., the same “description” of the LRMA that Frieze et al. use, which again ignores the  $O(kmn)$  time required to actually compute the approximate matrix from the description). Producing a full SVD from this output is non-trivial; see Algorithm 5.1 and Lemma 4. In particular, the approximate singular vectors from these methods are

not aligned in their subspace, in the sense that they do not successively capture a maximal amount of the input matrix magnitude. Because of this non-maximality, the  $\widehat{U}$  and  $\widehat{\Sigma}$  returned by the Drineas et al. methods fail to generate a  $\widehat{V} = A^T \widehat{U} \widehat{\Sigma}^{-1}$  with orthonormal columns, which is another critical feature of the true SVD. Many applications depend on both the maximality and orthogonality guarantees of a true SVD, and cannot tolerate this breaking of the SVD semantics.

Column-only and the analogous row-only sampling methods turn out to have good numerical stability, so most later work has favored these over joint row/column sampling. Empirical studies by Drineas et al. [40, 38] observe that even the tighter  $O(k/\epsilon^2)$  row-sample complexity bound is generally quite loose, i.e., targeted error levels can actually be achieved with many fewer samples than are required for the theoretical error guarantee. In [40], they suggest that approximation error could be checked in empirical Monte Carlo fashion against the input matrix, and the sample size increased until the error condition is satisfied. This idea was never pursued, most likely for three reasons: 1) the optimal error  $\|A - A_k\|_F^2$  is unknown, and therefore cannot be checked empirically, 2) even if it could be, the proposed idea would require full recalculation of the low-rank approximation subspace each time the sample size is increased, which becomes expensive, and 3) the additive nature of the error bound of Equation 42 is not very useful anyway. A related notion of empirical, iterative error checking, however, will be an essential ingredient in our approach, since our whole-matrix error criterion can be empirically checked and our method allows us to reuse the results of intermediate sample sizes without recomputation.

Achlioptas et al. [2, 1] present an alternative type of sampling method that performs randomized sparsification of the input matrix to speed standard  $k$ -rank SVD methods (e.g., orthogonal or Lanczos iteration). Standard  $k$ -rank SVD methods spend the bulk of their time computing matrix-vector products, which are computationally cheaper when the matrix is sparse. Achlioptas et al. show how a certain type of

randomized perturbation of the input matrix can increase sparsity without disturbing results in expectation. Computation time for the sparsified algorithm is at least  $O(M + kM' + kn^2)$ , where  $M$  is the number of non-zero entries in the input, and  $M'$  is the number of non-zero entries after sparsification. Error bounds are similar in form to Equation 42, for both Frobenius and spectral norms, with one subtlety: the additive error is proportional to a constant term to which the runtime is inversely proportional. The authors claim this method is comparable to the Drineas et al. column-sampling method in case of the Frobenius bound, and generally better in the case of the spectral bound.

### 5.3.2 Low-Rank Approximation with Relative Error Control

Deshpande and Vempala [31] provided the first Monte Carlo LRMA method with a relative error bound. Their bound takes the following form:

$$\|A - \widehat{A}\|_F^2 \leq (1 + \epsilon)\|A - A_k\|_F^2. \quad (43)$$

This bound is obtained by sampling in rounds, where rows are sampled in each round from the length-squared distribution of the *residual* of  $A$  relative to the span of the samples from previous rounds. The bound is achieved with  $O(k/\epsilon + k^2 \log k)$  samples, but each recalculation of the sampling distribution is expensive, resulting in  $O((k/\epsilon + k^2 \log k)M + (k^2/\epsilon^2 + k^3 \log k/\epsilon^2 + k^4 \log^2 k)(m + n))$  runtime, where  $M$  is the number of non-zero entries in  $A$  ( $M = mn$  in the dense case, making the runtime at least  $O(k^2 mn)$ ).

Friedland et al. [51] specified an LRMA method with no actual error guarantee but that attempts an *ad hoc* relative error termination criterion. Their algorithm refines a  $k$ -dimensional subspace through a series of iterations to capture an increasing amount of the input matrix magnitude  $\|A\|_F$ . Each iteration extends the subspace by orthonormalizing a few newly sampled rows, finds the SVD of  $A$ 's projection onto the extended subspace, then truncates the SVD basis back to rank  $k$ . This procedure

is guaranteed never to worsen the subspace, and when an iteration fails to improve the subspace by more than some  $\epsilon$ , the algorithm terminates. The method runs in  $O(kmn)$  time; however, this ignores the unbounded number of iterations. The major drawback is the lack of an error guarantee: the algorithm could easily sample a row set that happens to add very little improvement to its current subspace, resulting in premature termination. On the positive side, their method is capable of returning an actual SVD, albeit at the fixed low rank  $k$ . Empirical results show reasonable convergence to near-optimal error with the termination criterion turned off (i.e., the algorithm is allowed to run indefinitely and the input matrix is small enough that the approximation can be checked against the known exact answer). Speedups are somewhat disappointing though, reaching only 42 for a random  $8000 \times 200$  matrix. The idea of updating an intermediate subspace while checking an error condition, however, is a key one that we will leverage.

Finally, Sarlós describes a method based on random projections [125]. He shows that with  $\Theta(k/\epsilon)$  random combinations of rows one can produce a  $k$ -rank approximation satisfying:

$$\|A - \widehat{A}\|_F \leq (1 + \epsilon)\|A - A_k\|_F . \quad (44)$$

Note that this is on the norm, rather than the squared norm as in Equation 43. Runtime is  $O((\frac{k}{\epsilon}M + (m+n)k^2/\epsilon^2) \log \frac{1}{\delta})$ , with  $M$  the number of non-zero entries in  $A$  and  $1 - \delta$  the probability of success. No empirical studies have been performed, but, as with previous methods, the  $\Theta(k/\epsilon)$  bound is almost certainly loose and may contain significant hidden constants.

Additionally, some of these techniques have been applied or adapted to clustering [40, 54, 32, 33], matrix multiplication [41, 42], and collaborative filtering [43].

### 5.3.3 Summary of LRMA Sampling Techniques

Most of the LRMA algorithms feature a sampling method used to build up a subspace on which to project the input matrix with bounded error. Though it solves a different problem, our SVD method operates similarly, so these sampling methods are directly comparable to our tree-based approach. Three main sampling techniques are presented in the literature,<sup>1</sup> and we will discuss each from the perspective of iteratively sampling a set of points, updating a subspace to include their span, and continuing until the subspace captures the input matrix to within a desired error threshold. This is how our method works, and it is similar to the framework used by Friedland et al. [51]. Note that the overall guiding principle in sampling for subspace construction is that the subspace will be constructed most efficiently if each new sample point is a good representative of the points that are not yet well represented in the subspace.

#### 5.3.3.1 Length-Squared Sampling (LS)

In this scheme, rows are sampled with probabilities proportional to their squared lengths:  $p_i = \|A_{(i)}\|_F^2 / \|A\|_F^2$ . LS sampling was used in the seminal work of Frieze, Kannan, and Vempala [53], as well as in the follow-on work of Drineas et al. [40, 38, 39]. It is essentially an importance sampling scheme, assigning higher probabilities to rows that have more impact on the error objective  $\|A - \hat{A}\|_F^2$ , and is shown to minimize expected error among all fixed probability distributions. However, it has two important weaknesses. First, a row can have high norm while not being representative of other rows. Second, the distribution is non-adaptive, in that a point is equally likely to be drawn whether it is already well represented in the subspace or not. Both of these lead to wasted samples and needless inflation of the subspace rank.

---

<sup>1</sup>We exclude theoretical constructs such as the volume sampling technique of Deshpande and Vempala [31], which has no practical implementation, as well as redundant reformulations of equivalent sampling schemes.

### 5.3.3.2 Residual Length-Squared Sampling (RLS)

Residual length-squared sampling was introduced by Deshpande and Vempala [31]. RLS modifies the length-squared distribution after each subspace update by setting  $p_i = \|A_{(i)} - \Pi_V(A_{(i)})\|_F^2 / \|A - \Pi_V(A)\|_F^2$ , where  $\Pi_V$  represents projection onto the current subspace  $V$ . In other words, it samples from the LS distribution of the row *residuals* relative to the current subspace. By adapting to the residuals, this method avoids drawing samples that are already well represented in the subspace. Unfortunately, there is still nothing to enforce that any sample drawn will be representative of other rows with high residual length. Further, the updating of the residuals requires an expensive  $s$  passes through the matrix for every  $s$  samples that are added, which significantly limits practical utility.

### 5.3.3.3 Random Projections (RP)

Sarlós' random projection method [125] rounds out the list of sampling techniques. The idea is to sample random linear combinations of the rows, where the combination coefficients are random numbers, e.g., drawn from a standard Gaussian. This method is strong in one aspect where LS and RLS are weak — because all rows influence every sample, each sample is likely to represent a sizeable number of rows. A disadvantage, however, is that the combination coefficients are not informed by importance, e.g., a row of small length with a high coefficient can wash out many rows of great length whose coefficients are small. Further, each linear combination requires a full matrix pass, and the sampling distribution is non-adaptive, just as in basic LS. One could imagine doing random projections on residuals, but it would be costly, requiring two full matrix passes per sample.

Each of these sampling methods presents both strengths and weaknesses. Some of the weaknesses may have been overlooked or tolerated previously because the methods lend themselves to proving elegant bounds. Nonetheless, there is a clear opportunity

to combine aspects of these ideas with a more practical slant. We introduce a new sampling method based on a novel partitioning scheme called the *cosine tree*. Cosine tree sampling will be seen to dominate all previous sampling methods, and forms the foundation of the QUIC-SVD.

#### ***5.4 QUIC-SVD: Whole-Matrix SVD Approximation with Relative Error Control***

Rather than produce low-rank matrix approximations or even low-rank SVD approximations, we seek to solve a different but related problem: approximation of the whole-matrix SVD with relative error control. Table 5 summarizes the differences between our reformulated problem of whole-matrix SVD approximation and the problem of low-rank matrix approximation which was the focus of previous work. First, we produce a true SVD, with the proper diagonal  $\Sigma$  and orthogonal  $U$  and  $V$  whose columns capture successively maximal amounts of the input matrix magnitude within their respective subspaces. Contrast this with LRMA, which returns a low-rank matrix  $\hat{A}$  or an unaligned (non-successively-maximal)  $\hat{V}$  (or  $\hat{U}$ , but not both) and  $\hat{\Sigma}$  only. Second, we approximate the SVD of the full original matrix, rather than approximating relative to a low-rank version. Third, because our approximations are with respect to the full matrix, our error bound is also relative to the full matrix, rather than being additive or relative to a low-rank version of the matrix.

These distinctions are critical for numerous applications, such as solving linear systems, least-squares problems, or anything requiring the matrix pseudoinverse, such as Gaussian process regression or kernel ridge regression. They are also important because they preserve the semantics of the SVD, which are frequently and sometimes implicitly relied upon in algorithms making use of the SVD. Another reason to address the whole-matrix SVD is that in many applications (e.g., PCA) we need as many singular vectors as are necessary to capture a specific fraction of the magnitude/variance of the input matrix. We thus need our approximation to have some

**Table 5:** Distinctions between whole-matrix SVD approximation and low-rank matrix approximation.

Whole-matrix SVD approximation	Low-rank matrix approximation
True SVD: $U$ , $\Sigma$ , and $V$	$\hat{A}$ or unaligned $\hat{V}$ & $\hat{\Sigma}$ only
Address full-rank matrix	Fixed low-rank $k$
Full-rank relative error bound	$k$ -rank error bound, additive or relative

minimal rank; however, we cannot know the required rank without first computing the full or nearly full SVD! We therefore have no way, *a priori*, of knowing what value of  $k$  to use in computing a fixed  $k$ -rank SVD. From a practical perspective, then, it is often more useful to approximate the whole-matrix SVD than one of fixed rank  $k$ .<sup>2</sup>

Using the true whole-matrix SVD as our target also makes the approximation task significantly harder than LRMA. Specifically, we target the following whole-matrix relative squared error bound:

$$\|A - \hat{A}\|_F^2 \leq \epsilon \|A\|_F^2, \quad (45)$$

where  $A$  is the input matrix and  $\hat{A} = \hat{U}\hat{\Sigma}\hat{V}^T$  is the matrix reconstructed from our approximate SVD. It is crucial that any error bound have a relative form in order to be meaningful. Further, Frobenius-norm bounds such as this are more powerful than the spectral-norm bounds used in some related work. Since the approximation target is now the known  $A$  instead of an unknown  $A_k$ , we can check error empirically (exactly or approximately) against it. This enables us to formulate an algorithm that iteratively builds up the approximation, checks error, and continues building only if the error bound is not yet satisfied. Because we can terminate immediately once the desired error tolerance is reached, we can adaptively minimize the number of samples and therefore maximize computational speed. Contrast this with the low-rank approximation methods that mostly produce their approximations as a one-off

---

<sup>2</sup>It is understandable that work with a primarily theoretical focus would ignore the case of approximating the full matrix, since its exact error can be trivially checked against the input matrix, which obviates the derivation of error bounds.

**Table 6:** Distinctions between subspace construction in QUIC-SVD and previous LRMA methods.

<b>QUIC-SVD</b>	<b>Previous LRMA Methods</b>
Iterative buildup with tight empirical error control	Fixed-size computation with loose error bound
Adaptive sample size minimization	Fixed <i>a priori</i> sample size (loose)
Cosine tree sampling	Various sampling schemes

computation with a fixed, over-large sample size. Deshpande et al. do an incremental build-up and adapt the sampling distribution, but they do not adapt the sample size, which remains fixed. Friedland et al. also build their approximation incrementally, but have no error bounds. None of them use empirical error control.

Thus, in addition to solving a different problem, the QUIC-SVD is also distinct from previous work in the way that it discovers a suitable subspace and controls for relative error. Table 6 summarizes the differences in approach. Key points include: 1) iterative buildup with fast empirical error checking, vs. pre-determined sampling rounds based on loose error bounds; 2) adaptive minimization of sample size based on empirical error checks, vs. fixed *a priori* sample sizes determined by loose bounds; and 3) a highly efficient sampling scheme based on cosine trees, vs. various less efficient sampling schemes.

#### 5.4.1 Cosine Trees

QUIC-SVD operates by building up a subspace to capture the input matrix  $A$  to a desired degree of fidelity, at which point it extracts the SVD of  $A$ 's projection into the subspace. For efficiency, the subspace is built via a sampling procedure, and for maximum efficiency we need to minimize the rank of the final subspace. We therefore require the most rank-efficient of sampling procedures. The LRMA literature provides several sampling options, but as noted above, each option has significant drawbacks.

The cosine tree enables a new sampling procedure for subspace discovery. It

---

**Algorithm 5.2** Cosine tree construction.

---

CTNODE

**Input:**  $A \in \mathbb{R}^{m \times n}$ **Output:** cosine tree node containing the rows of  $A$ 

1.  $N \leftarrow$  new cosine tree node
2.  $N.A \leftarrow A$
3.  $N.splitPt \leftarrow \text{ROWSAMPLELS}(A)$
4. **return**  $N$

CTNODESPLIT

**Input:** cosine tree node  $N$ **Output:** left and right children obtained by cosine-splitting of  $N$ 

1. **for each**  $N.A_{(i)}$ , compute  $c_i = |\cos(N.A_{(i)}, N.splitPt)|$
  2. **if**  $\forall i, c_i = 1$ , **return** *null*.
  3.  $c_{max} = \max\{c_i | c_i < 1\}$ ,  $c_{min} = \min\{c_i\}$
  4.  $A_l \leftarrow []$ ,  $A_r \leftarrow []$
  5. **for**  $i = 1$  to  $N.size$ 
    - (a) **if**  $c_{max} - c_i \leq c_i - c_{min}$ ,  $A_l \leftarrow \begin{bmatrix} A_l \\ N.A_{(i)} \end{bmatrix}$
    - (b) **else**  $A_r \leftarrow \begin{bmatrix} A_r \\ N.A_{(i)} \end{bmatrix}$
  6. **return** CTNODE( $A_l$ ), CTNODE( $A_r$ )
- 

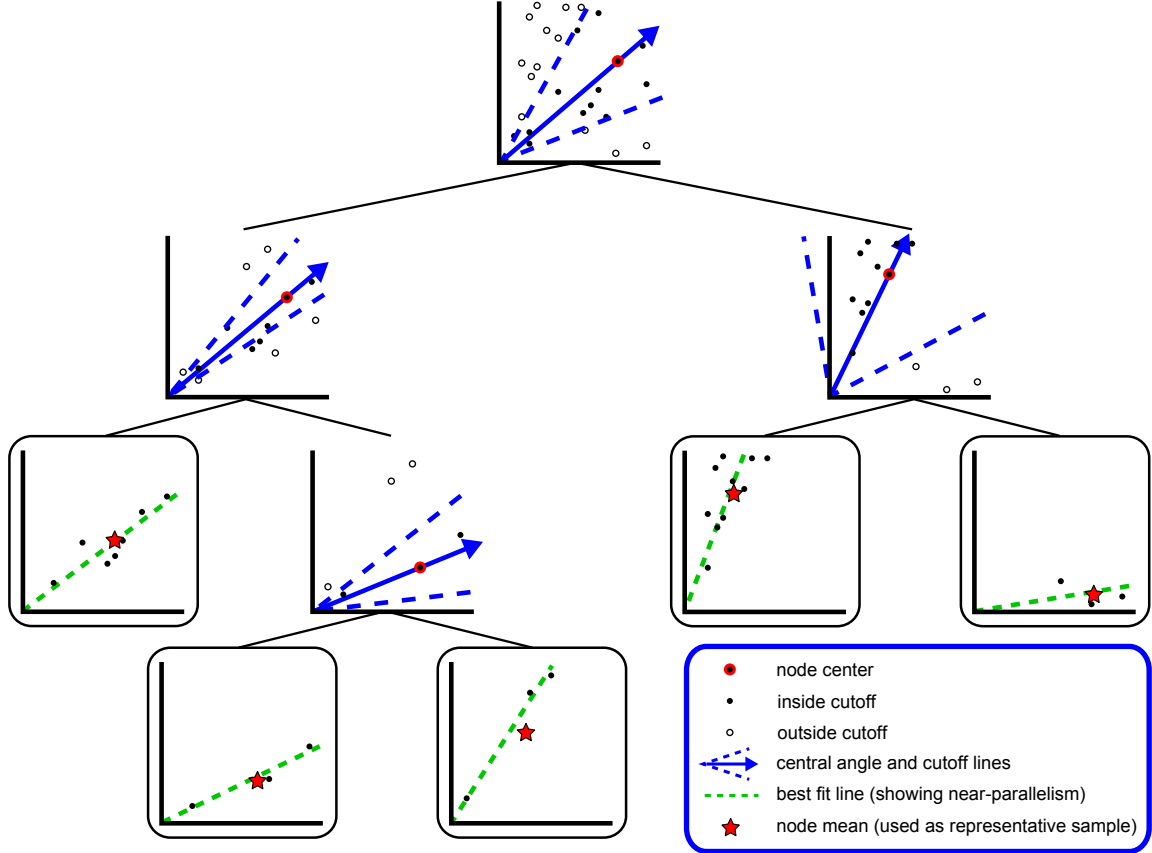
possesses the strongest features of all previous LRMA sampling procedures: computational ease, adaptivity to intermediate sampling results, and representation of more than one row at a time. Because it combines these strengths, it empirically dominates previous sampling methods in terms of sample efficiency (see Section 5.5.1), thus making it the right sampling procedure for our SVD approximation algorithm.

The procedure for constructing a cosine tree is specified in Algorithm 5.2. The essential concept is to sort the rows of the matrix into groups within which all rows are

close to lying on the same subspace. Once we achieve this, we need only guarantee good representation of one representative from each group, and the rest will also be well represented. Mutual closeness is gauged by absolute cosines. We illustrate this idea with an example in Figure 24. In this example, the cosine tree is over the rows of a two-column matrix, which we illustrate as points in two-dimensional space. Starting at the root node, a split point is chosen by sampling from the LS distribution, which is a cheap source of information about the relative significance of each point. The remaining points are then sorted in descending order of their absolute cosine relative to the split point. We split the sorted list at the point where the cosines becomes closer to the low end than the high end of the list. This amounts to fixing an angular cutoff around the line through the split point, as illustrated by the dashed lines in Figure 24. The idea is that the points within the angular cutoff are closer to lying on a common subspace than the points outside the cutoff, and therefore can be grouped for a compact-but-high-fidelity representation. Representation is provided by the centroid of each node, which captures some influence from each point in the node. A subspace being constructed would be expanded so that its span covered the representative point of each newly expanded node.

The splitting procedure repeats recursively, as within-cutoff points go to the left child and without-cutoff points go to the right child. A node with all points parallel or antiparallel is not split. As shown in the figure, this eventually results in frontier nodes whose points are close to lying along the same line. In the QUIC-SVD algorithm, the expansion order is prioritized by the residual error of each frontier node relative to the current subspace.

Why is this helpful for subspace discovery? The ideal subspace discovery algorithm would oracularly choose as samples the singular vectors  $v_i$ . Each  $v_i$  is precisely the direction that, added to the subspace spanned by the previous singular vectors, will capture the most residual magnitude over all rows of the matrix. Because cosine



**Figure 24:** A cosine tree in two dimensions.

tree expansion is prioritized by the residual error of the frontier nodes, sampling is always focused on the areas where the most residual magnitude remains to be captured. As cosine-based splitting guides the nodes toward having nearly parallel rows, the residual magnitude of each node is increasingly likely to be mostly captured along the direction of the node centroid. Expanding a subspace in the direction of the highest-priority node centroid is therefore a good guess as to the direction that will capture a maximal amount of the residual matrix magnitude. Thus, cosine tree sampling approximates the ideal of oracularly sampling the true singular vectors.

Note that the cost of a single split is  $O(mn)$  to compute  $O(m)$  cosines at a cost of  $O(n)$  each. Also, the length-squared distribution need only be computed once, at a cost of  $O(mn)$ . Expanding a cosine tree to  $k$  frontier nodes therefore takes  $O(kmn)$  time ( $k - 1$  splits at  $O(mn)$  each plus the one-time  $O(mn)$  for the LS distribution).

### 5.4.2 QUIC-SVD

We now specify an algorithm, QUIC-SVD, for leveraging cosine trees to construct a whole-matrix SVD approximation with relative error control. Algorithm 5.3 lists the procedure. We provide a schematic illustration in Figure 25. The algorithm forms a cosine tree on the input matrix, then iteratively 1) splits the node with highest error, 2) expands a growing subspace basis to span the right child’s mean, and 3) checks the projection error of the input matrix onto the subspace. Once the projection error is low enough, subspace construction terminates and the EXTRACTSVD algorithm is invoked to return the SVD of the projection of the input matrix onto the subspace. The error guarantee provided by QUIC-SVD is formalized as Theorem 11.

**Theorem 11.** *Given a matrix  $A \in \mathbb{R}^{m \times n}$  and a relative error threshold  $\epsilon \in [0, 1]$ , the algorithm QUIC-SVD returns an SVD  $U, \Sigma, V$  such that  $\hat{A} = U\Sigma V^T$  satisfies  $\|A - \hat{A}\|_F^2 \leq \epsilon \|A\|_F^2$ .*

*Proof.* We need to establish 1) that the algorithm terminates, and 2) that it does so with an SVD satisfying the stated relative error bound. We first show termination. The algorithm proceeds by expanding the nodes of a cosine tree in an order prioritized by the residual squared error of each node’s projection onto the current subspace basis  $\hat{V}$  at the time the node is created. Each newly-created right child has the mean of its rows added to the basis  $\hat{V}$  after being orthonormalized relative to  $\hat{V}$  (line 4(d)). The splitting process is iterated until either  $sqErr < \epsilon \|A\|_F^2$  or  $Q.isEmpty()$ . If the first condition is ever satisfied, the algorithm terminates. If not, after  $m - 1$  splits we have  $m$  nodes, so each node contains only a single row and will eventually be popped from the queue with no insertion of child nodes (line 4(a-b)). The queue will therefore become empty, triggering the second loop-termination condition. Thus, the algorithm is guaranteed to terminate.

We now show the error bound is satisfied in either termination condition. In

---

**Algorithm 5.3** QUIC-SVD, a whole-matrix SVD approximation algorithm with relative error control.

---

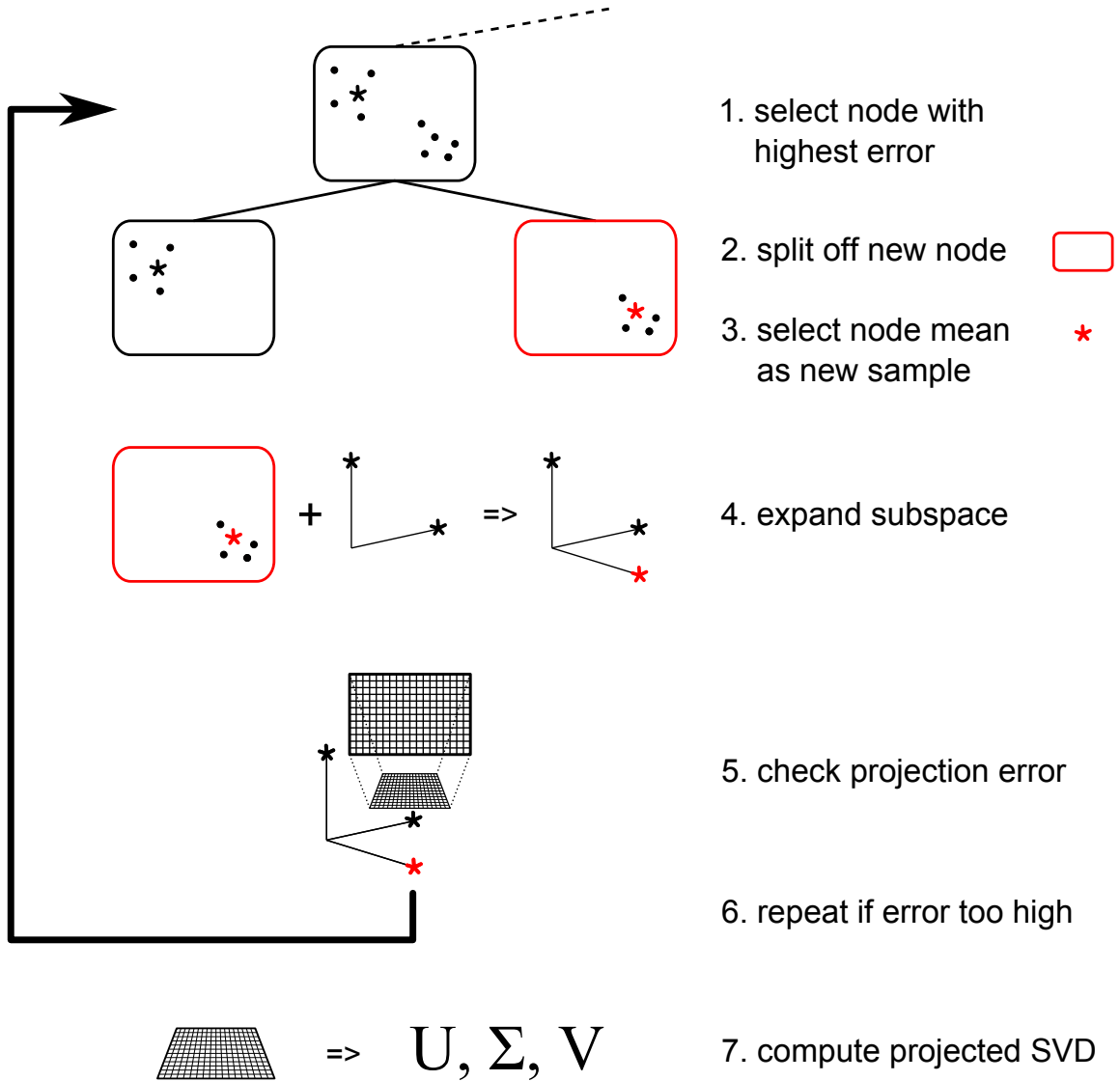
QUIC-SVD

**Input:**  $A \in \mathbb{R}^{m \times n}$ ,  $\epsilon \in [0, 1]$

**Output:** an SVD  $U, \Sigma, V$  s.t.  $\hat{A} = U\Sigma V^T$  satisfies  $\|A - \hat{A}\|_F^2 \leq \epsilon \|A\|_F^2$

1.  $\hat{V} \leftarrow []$ ,  $A\hat{V} \leftarrow []$ ,  $sqErr \leftarrow \|A\|_F^2$
  2.  $N_{root} \leftarrow \text{CTNODE}(A)$
  3.  $Q \leftarrow \text{EMPTYPRIORITYQUEUE}()$ ;  $Q.insert(N_{root}, 0)$
  4. **while**  $sqErr > \epsilon \|A\|_F^2$  and  $!Q.isEmpty()$ 
    - (a)  $N \leftarrow Q.pop()$ ;
    - (b) **if**  $N.size == 1$ , **continue**
    - (c)  $N_l, N_r \leftarrow \text{CTNODESPLIT}(N)$
    - (d)  $\mu_r \leftarrow N_r.rowMean$ ;  $v_r \leftarrow \text{unitize}(\mu_r - \mu_r \hat{V} \hat{V}^T)$
    - (e)  $\hat{V} \leftarrow [\hat{V} \ v_r]$ ;  $A\hat{V} \leftarrow [A\hat{V} \ Av_r]$  // add next column to  $\hat{V}$  and  $A\hat{V}$
    - (f)  $sqErr \leftarrow sqErr - \|Av_r\|_F^2$
    - (g)  $Q.insert(N_l, \|N_l.A\|_F^2 - \|(N_l.A)\hat{V}\|_F^2)$  // prioritize by residual error
    - (h)  $Q.insert(N_r, \|N_r.A\|_F^2 - \|(N_r.A)\hat{V}\|_F^2)$
  5. **return**  $\text{EXTRACTSVD}(A, \hat{V}, A\hat{V})$  // pass  $A\hat{V}$  to avoid recomputation
- 

the case of termination due to  $Q.isEmpty()$ , the subspace basis  $\hat{V}$  will have been expanded until it spans all rows of  $A$ .  $\text{EXTRACTSVD}$  (line 5) will therefore return the exact SVD of  $A$ , which trivially satisfies the theorem's error guarantee. Now consider the remaining case of termination due to  $sqErr \leq \epsilon \|A\|_F^2$ . We have already established that  $\text{EXTRACTSVD}$  (line 5) returns the SVD of  $A$ 's projection onto  $\hat{V}$  (see Lemma 4). Thus, the SVD returned by QUIC-SVD reconstructs to  $\hat{A} = A\hat{V}\hat{V}^T$ , where  $\hat{V}$  is the final subspace basis after termination of the main loop (line 4). We therefore need only show that the terminal  $\hat{V}$  satisfies  $\|A - A\hat{V}\hat{V}^T\|_F^2 \leq \epsilon \|A\|_F^2$ . Because  $A\hat{V}\hat{V}^T$  is an orthogonal projection, the Pythagorean theorem implies that  $\|A - A\hat{V}\hat{V}^T\|_F^2 = \|A\|_F^2 - \|A\hat{V}\hat{V}^T\|_F^2$ . Changing coordinate systems does not affect



**Figure 25:** Schematic illustration of the QUIC-SVD algorithm.

squared lengths, so  $\|A\widehat{V}\widehat{V}^T\|_F^2 = \|A\widehat{V}\|_F^2$ . The variable  $sqErr$  is initialized to  $\|A\|_F^2$  (line 1), and as each new column  $v_r$  is added to  $\widehat{V}$ ,  $sqErr$  is decreased by  $\|Av_r\|_F^2$ . Thus, at the start of each loop,  $sqErr = \|A\|_F^2 - \|A\widehat{V}\|_F^2$ , which, as we have shown, is equal to  $\|A - A\widehat{V}\widehat{V}^T\|_F^2$ . The  $sqErr \leq \epsilon\|A\|_F^2$  termination condition therefore implies  $\|A - A\widehat{V}\widehat{V}^T\|_F^2 \leq \epsilon\|A\|_F^2$ , which is all we need for the theorem to hold.  $\square$

#### 5.4.2.1 Running Time Analysis

Let the rank of the final subspace  $\widehat{V}$  be  $k$ . The running time analysis for QUIC-SVD is as follows:

1.  $O(kmn)$  for expanding the cosine tree to  $k$  nodes (line 4(c)), and for  $k$  computations of  $Av_r$  (line 4(e)).
2.  $O(k^2n)$  for orthonormalizations (line 4(d)).
3. No significant contribution from lines 4(g)-(h) because we track the projection error of each row incrementally as we compute the product  $Av_r$ ; thus,  $\|(N_{l/r}.A)\widehat{V}\|_F^2$  is already available.  $\|N_{l/r}.A\|_F^2$  has also already been computed in generating the length squared distribution for  $N_{l/r}.A$ .
4. Since we pass  $A\widehat{V}$  as an argument, only  $O(k^3)$  for EXTRACTSVD (line 5).

The overall runtime is therefore  $O(kmn)$ . Note that the  $O(kmn)$  contribution from expanding the cosine tree has a very small hidden constant, as most nodes below the root have far fewer than  $m$  points. The incremental multiplication of  $Av_r$  to produce the columns of  $A\widehat{V}$  is therefore the most costly operation by far. Note that the computation of  $Av_r$  accomplishes several tasks simultaneously: 1) computing the projection onto each new basis, 2) updating the global projection error, and 3) updating the per-row projection error to enable per-node error prioritization. Hitting these three birds with a single computational stone is crucial to the algorithm's speed.

But what of the final subspace rank  $k$ ? It depends both on the spectral structure of the matrix and the sampling efficiency of the prioritized cosine tree. The  $O(kmn)$  runtime means we are greatly benefited by a *compact* subspace — our error bound requires the subspace  $\widehat{V}$  to capture a threshold amount of the input matrix magnitude, and we favor subspaces that do so with minimal rank. The exact SVD gives us a lower bound on the rank required to achieve a given error threshold  $\epsilon$ . Recall that  $V_k$  from

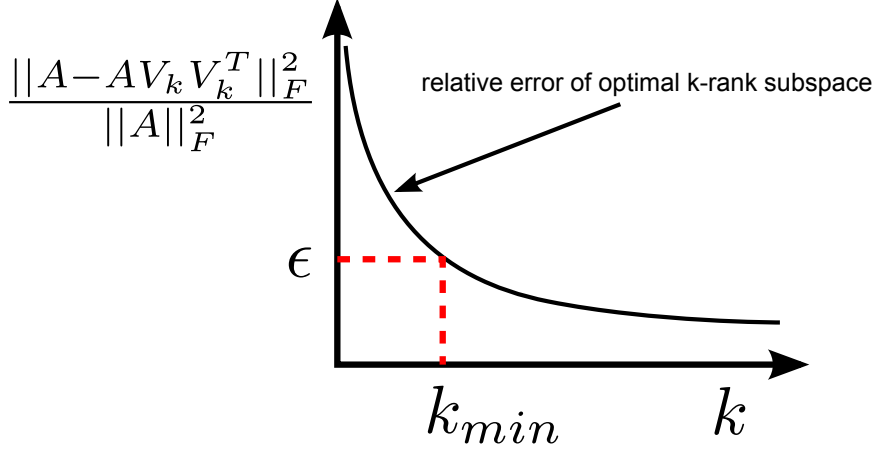
the  $k$ -truncated exact SVD captures the largest possible amount of  $A$ 's magnitude among all  $k$ -rank subspaces. Let  $k_\epsilon$  be defined as follows:

**Definition 4.**  $k_\epsilon$  is the smallest rank  $k$  such that the  $k$ -truncated SVD row basis  $V_k$  satisfies  $\|A - AV_kV_k^T\|_F^2 \leq \epsilon\|A\|_F^2$ .

Figure 26 illustrates how  $k_\epsilon$  is determined by the matrix spectrum. Since our  $\widehat{V}$  at rank  $k$  can do no better than  $V_k$ , it follows that QUIC-SVD will always terminate with  $k \geq k_\epsilon$ . In general this lower bound on  $k$  is not tight, since we have no way to sample the singular vectors of  $V_{k_\epsilon}$  directly. However,  $k_\epsilon$  provides an important link between the  $O(kmn)$  runtime of our algorithm and the spectral structure of the input matrix. This can be seen from the fact that  $\|AV_kV_k^T\|_F^2 = \sum_i \sigma_i^2$ , which implies that  $\|A - AV_kV_k^T\|_F^2 = \|A\|_F^2 - \|AV_kV_k^T\|_F^2 = \sum_{i=1}^\rho \sigma_i^2 - \sum_{i=1}^k \sigma_i^2 = \sum_{i=k+1}^\rho \sigma_i^2$ . Rearranging this yields an alternate definition:  $k_\epsilon$  is the smallest rank  $k$  such that  $\sum_{i=1}^k \sigma_i^2 \geq (\frac{1}{\epsilon} - 1) \sum_{i=k+1}^\rho \sigma_i^2$ . In other words,  $k_\epsilon$  is the tipping point at which the sum of the top  $k$  squared singular values becomes greater than an  $\epsilon$ -dependent multiple of the sum of the bottom  $\rho - k$  squared singular values. Clearly, matrices whose spectra are more concentrated in the top singular values will have smaller values of  $k_\epsilon$ . Our experiments will show that real datasets tend to show such concentration, and it is precisely these cases in which QUIC-SVD will give greatest acceleration.

Additionally, the speedup of QUIC-SVD over exact SVD can be written as  $O(\frac{mn^2}{kmn}) = O(\frac{n}{k})$ . This implies that among matrices with the same number of entries ( $mn$ ), QUIC-SVD will show greatest speedup for those that are closest to square (largest  $n$ ). Nonetheless, as we will see, QUIC-SVD can still give substantial speedup even in the case of thin matrices and matrices with broad spectra.

Finally, we note that while our discussion of QUIC-SVD has been in terms of dense matrices, the algorithm is certainly applicable to sparse matrices as well. Not all sparse matrices, however, will see significant speedup with QUIC-SVD. This is because QUIC-SVD performs best when there is significant linear structure in the



**Figure 26:**  $k_\epsilon$  is determined by the spectrum of the matrix, as represented here by a graph of the relative squared error of  $AV_k V_k^T$  vs.  $k$ .

data, such that a lower-dimensional subspace can capture much of the input matrix magnitude. In the sparse case, however, rows tend to be mostly orthogonal to one another, such that a nearly full-rank subspace is required for low-error approximation (i.e., the spectrum is broad and  $k_\epsilon$  is high). QUIC-SVD still works but is less advantageous in such cases. Nonetheless, for any sparse matrix with sufficient lower-dimensional linear structure, QUIC-SVD will provide good speedup. For this reason, we do not restrict our scope to the dense case.

### 5.5 Empirical Results

We present three phases of experimentation. First, we compare the efficiency of prioritized cosine tree sampling to that of the various sampling methods from the low-rank matrix approximation literature. Cosine tree sampling shows empirical dominance across a battery of real datasets. Second, we measure the speed and error performance of QUIC-SVD on a variety of medium-sized dataset matrices (millions to tens of millions of entries) of differing shapes and spectral structures. Finally, we test the scalability of QUIC-SVD through timing experiments on a set of large-scale matrices containing hundreds of millions to billions of entries.

### 5.5.1 Cosine Trees vs. Previous Sampling Methods

Because of the  $O(kmn)$  runtime of QUIC-SVD, it is imperative that its sampling method be efficient in generating high-accuracy subspaces with compact rank. It should be clear that prior LRMA sampling techniques could be substituted for the cosine tree technique used in QUIC-SVD. What justifies our use of cosine trees? We demonstrate in this section that cosine tree sampling empirically is dominantly more rank-efficient than prior LRMA sampling methods. Because the final rank  $k$  is the critical determinant of QUIC-SVD runtime, rank-efficiency is paramount. This must be balanced, however, against the computational cost of the various sampling procedures. The residual length-squared and random projection approaches require a full matrix pass per sample, and are therefore not only less rank-efficient but significantly more expensive than cosine tree sampling, in which each sample requires only a pass through the selected node. (Note that this disparity in computational cost is also the reason we use cosine trees rather than Arnoldi iteration, which requires an expensive matrix-vector product per expansion of the subspace — such products are cheap for the sparse matrices targeted by Arnoldi iteration, but we are targeting the general case). Simple length-squared or uniform sampling are strictly cheaper than cosine tree sampling, but they are less rank-efficient by a large margin, which washes out the smaller per-sample cost. Thus, the dominant rank-efficiency and reasonable per-sample cost of cosine tree sampling make it the right technique for QUIC-SVD.

To demonstrate how cosine tree sampling is an improvement over the state of the art, we give a series of plots showing the average relative squared error versus subspace rank on a series of real dataset matrices for the following sampling procedures: cosine tree (CT), length-squared (LS), residual length-squared (RLS), random projections (RP), and uniform (with replacement) (UNI). As a baseline we show the optimal error line of the true SVD subspace at each rank (Opt). We use a set of medium-sized matrices (millions to tens of millions of entries) for this set of experiments so

**Table 7:** Descriptions of the six matrices used for medium-scale experiments, ranging from 3.6 million to 53 million entries in size.

<b>name</b>	<b>size</b>	<b>description</b>
mars	$2,000 \times 1,783$	image of the Martian surface
madelon-kernel	$2,000 \times 2,000$	Gaussian kernel matrix derived from NIPS 2003 Workshop on Feature Extraction
declaration	$4,656 \times 3,923$	high-res scan of US Decl. of Independence
mnist	$70,000 \times 754$	a standard handwritten character recognition dataset
arcene	$10,000 \times 900$	from NIPS 2003 Workshop on Feature Extraction
galaxy	$10,000 \times 3,840$	spectra of 10,000 galaxies from the Sloan Digital Sky Survey

that the exact SVD is feasible to compute. These matrices are described in Table 7. Figures 27–32 show the results. All sampling results are averaged over five runs.

Cosine tree sampling dominates the other methods in that its error line is always lowest. The separation is usually quite large, with two exceptions: 1) arcene, where LS, RLS, and RP closely track CT, and 2) galaxy, where RLS is somewhat close to CT. It is particularly striking how close the CT line is to the true SVD line for three out of six matrices (mars, madelon-kernel, and mnist). No other sampling procedure comes close to such performance. This result is remarkable given how much cheaper the cosine tree is than exact SVD or the next-best competitors, RLS and RP.

It is also worth noting how the sampling error lines follow the general shape of the exact SVD line. This speaks to the way in which spectral structure affects the rank arrived at in the subspace buildup of QUIC-SVD. Specifically, the lower bound  $k_\epsilon$  for any  $\epsilon$  can be found by tracing the horizontal line from  $\epsilon$  on the vertical axis to its intersection with the Opt line. The *actual*  $k$  achieved by cosine tree sampling is found at the intersection with the CT line. Clearly,  $k \geq k_\epsilon$  in all cases. While  $k$  is not directly determined by the spectrum as  $k_\epsilon$  is, it is nonetheless governed by the general shape of the spectrum: if the Opt line falls sharply,  $k$  is small just as  $k_\epsilon$  is small, while a flatter Opt line results in a larger  $k$  just as it results in a larger  $k_\epsilon$ .

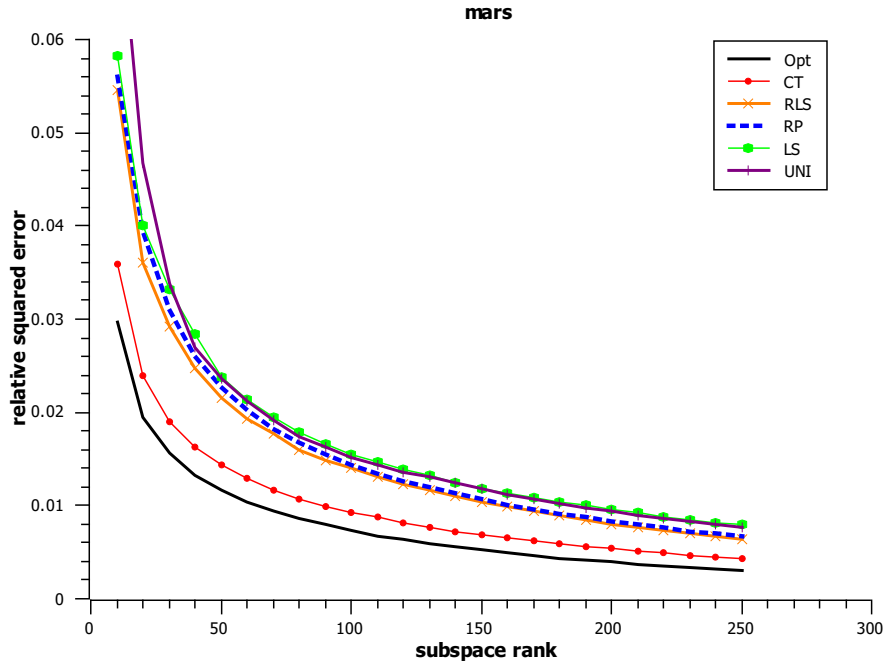


Figure 27: Relative squared projection error vs. subspace rank for mars.

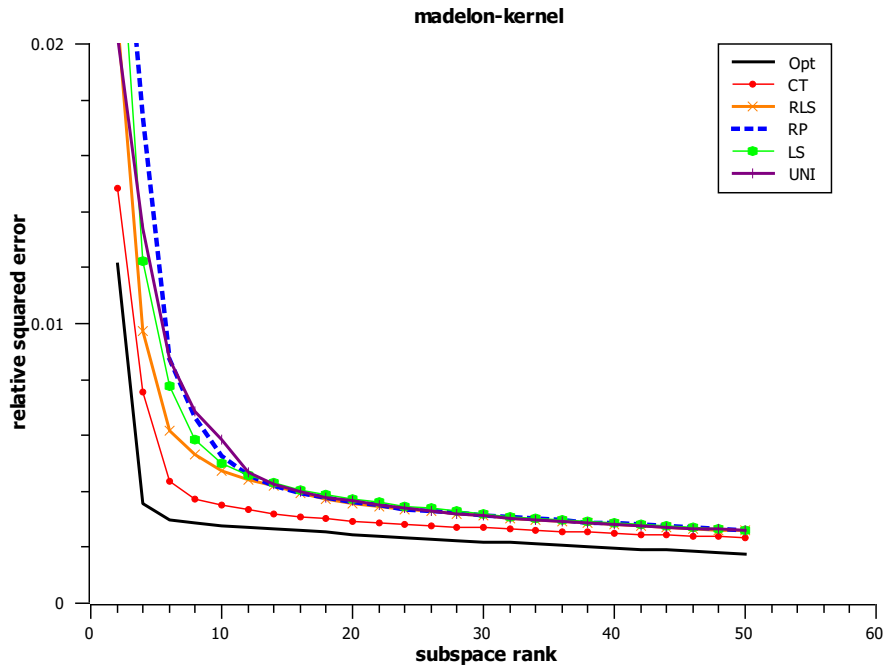


Figure 28: Relative squared projection error vs. subspace rank for madelon-kernel.

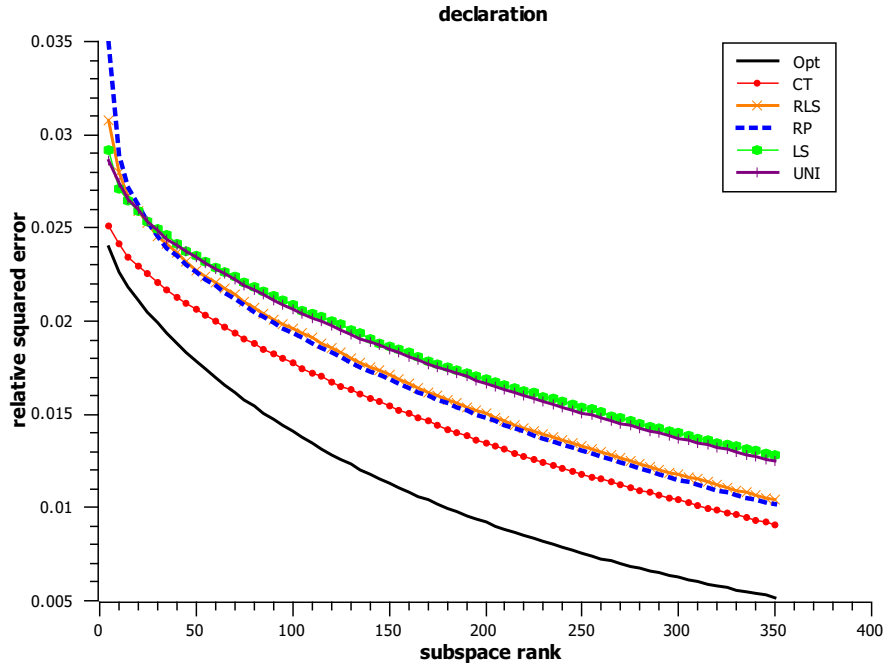


Figure 29: Relative squared projection error vs. subspace rank for declaration.

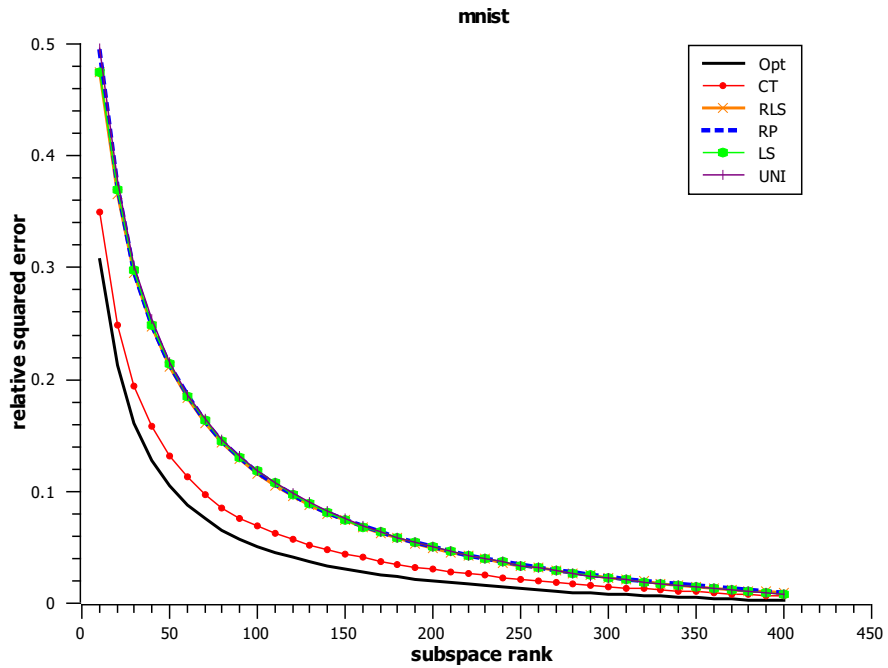


Figure 30: Relative squared projection error vs. subspace rank for mnist.

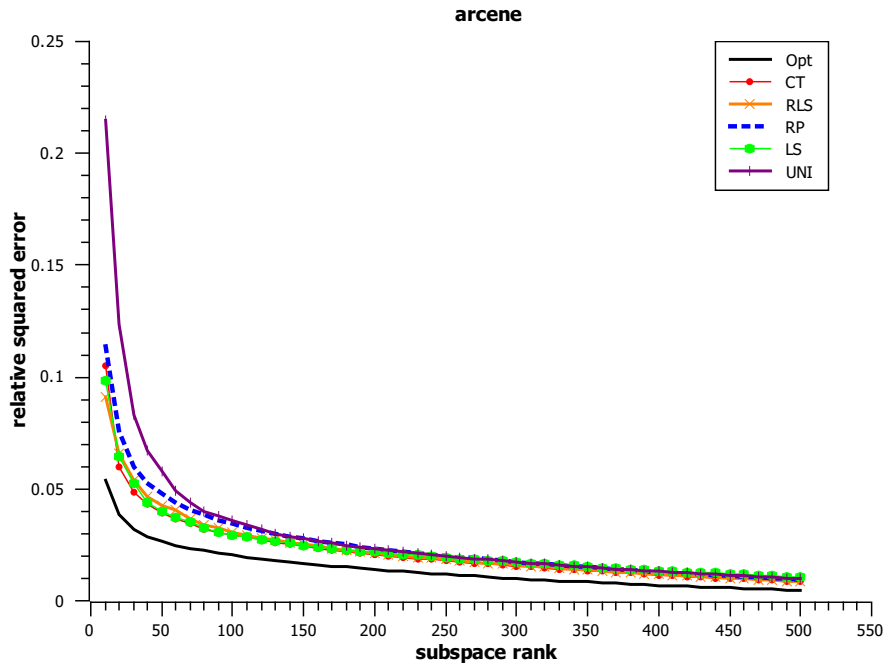


Figure 31: Relative squared projection error vs. subspace rank for arcene.

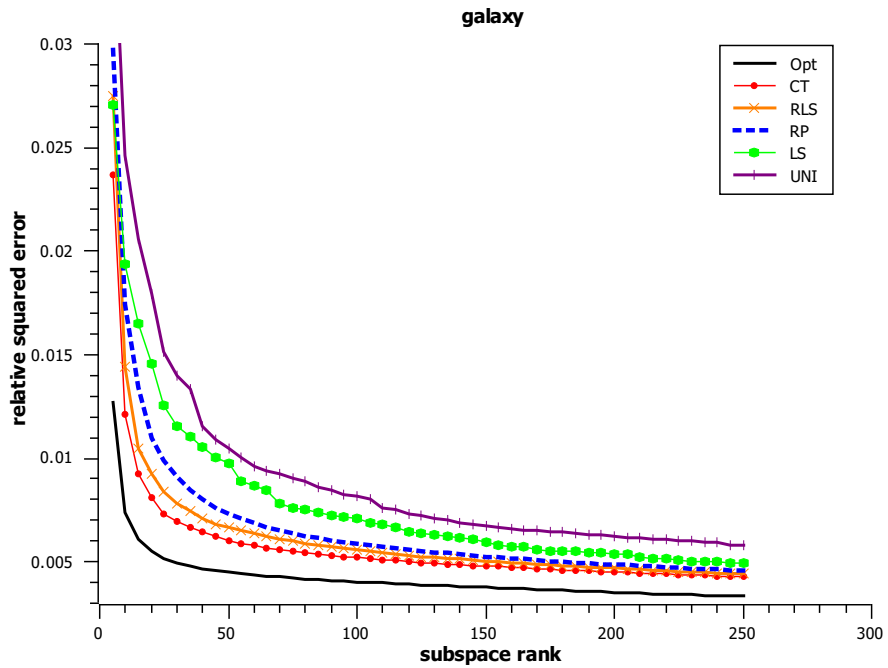


Figure 32: Relative squared projection error vs. subspace rank for galaxy.

### 5.5.2 Error Control and Speedup on Medium-Scale Matrices

With the foregoing results establishing the effectiveness of prioritized cosine tree sampling, we now turn to the composite performance of the QUIC-SVD. Our goal is twofold: 1) to provide empirical verification of the QUIC-SVD error guarantee, and 2) to gauge the speedup provided by QUIC-SVD over exact SVD on a variety of real datasets. We ran QUIC-SVD on the six medium-scale matrices described in Table 7. Though various in shape, these matrices represent the scale of millions to tens of millions of entries. We ran QUIC-SVD at a range of  $\epsilon$  error tolerances. Each run was replicated five times. We also ran an exact SVD<sup>3</sup> to get a baseline runtime for comparison. For verification of error control, we report the minimum and maximum relative squared errors (as a fraction of the target  $\epsilon$ ) observed across all runs at all error levels on each matrix. These results are listed in Table 8. For runtime performance, we show for each matrix a plot of speedup (over exact SVD) vs. error tolerance  $\epsilon$  in Figures 33–38.

The main thing to note about the error performance is that it never exceeds the target  $\epsilon$ , as indicated by the fact that none of the maximum fraction-of- $\epsilon$  relative errors is greater than 1. The error guarantee of Theorem 11 is thus demonstrated empirically. More interestingly, the runtime performance shows significant speedups over exact SVD, ranging from the tens to the low tens of thousands at error levels of 0.01–0.1. Note particularly the effect of squareness. The squarer matrices (mars, madelon-kernel, declaration, galaxy) show significantly higher speedups, from the low hundreds to the mid thousands and even as high as 15,000 as  $\epsilon$  increases. The thinner matrices, on the other hand, start in the tens and rise no higher than  $\sim 1,000$ . This is in accord with the order  $O(\frac{n}{k})$  speedup expression given in Section 5.4.2.1.

Thus, overall, we see speedups on the order of  $10$ – $10^4$  for the medium scale. In

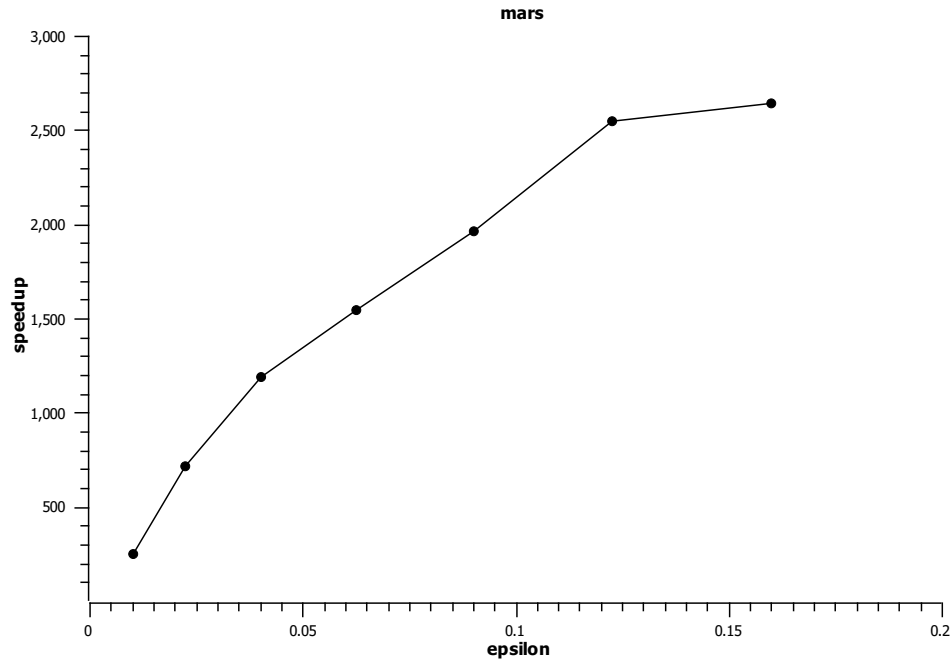
---

<sup>3</sup>The SVD we used is based on that of LINPACK, and is nearly identical to that used in LAPACK or MATLAB.

**Table 8:** Minimum and maximum relative squared error (as a fraction of  $\epsilon$ ), across all runs at all error levels for each medium-scale matrix.

<b>matrix</b>	<b>min</b>	<b>max</b>
mars	0.688	0.999
madelon-kernel	0.566	0.999
declaration	0.679	0.999
mnist	0.965	0.999
arcene	0.933	0.999
galaxy	0.767	0.999

absolute terms, the largest of these speedups reduces a 7.4 hour exact runtime to 1.7 seconds at 4% squared error, 4.3 seconds at 1% squared error, or 8.9 minutes at 0.25% squared error (galaxy matrix). Even the matrices with lower speedups still showed impressive absolute reductions, such as from 3.9 hours to 2.4 minutes at 4% error for mnist. Such acceleration represents a qualitative leap into a new speed regime.



**Figure 33:** Speedup vs. error tolerance for mars.

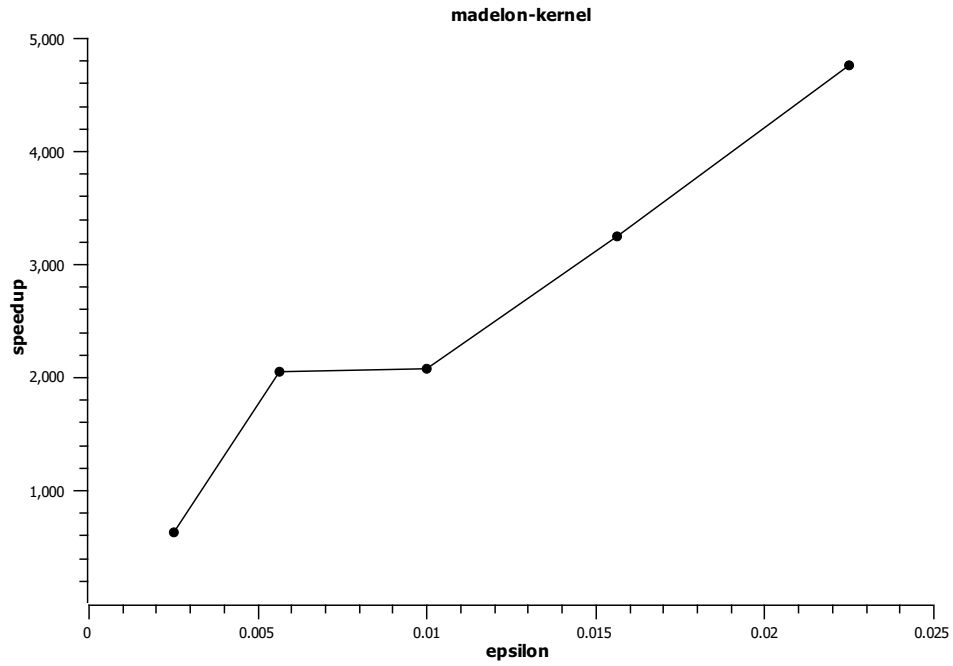


Figure 34: Speedup vs. error tolerance for madelon-kernel.

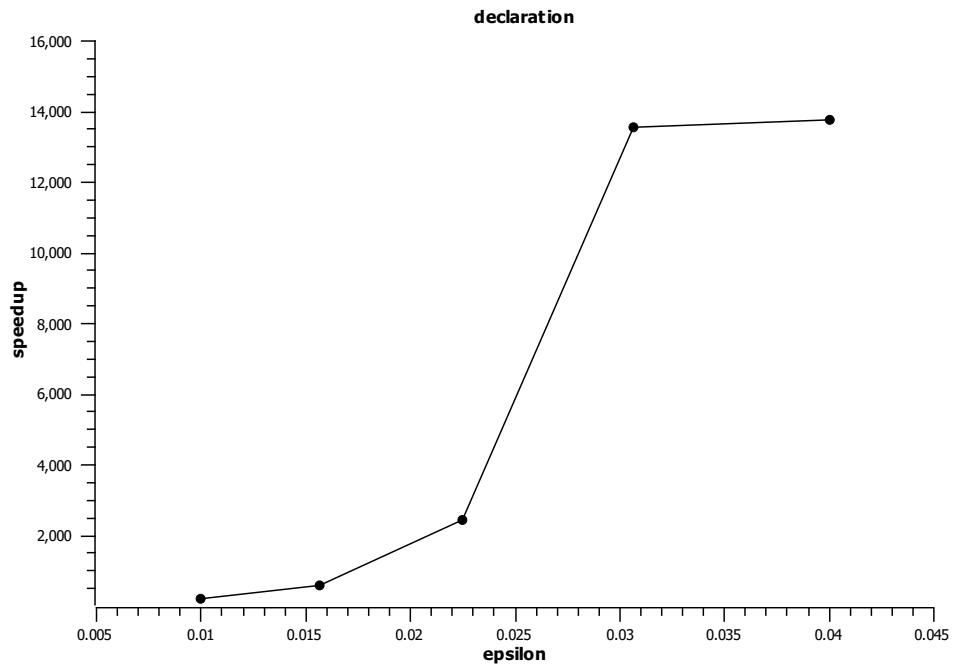


Figure 35: Speedup vs. error tolerance for declaration.

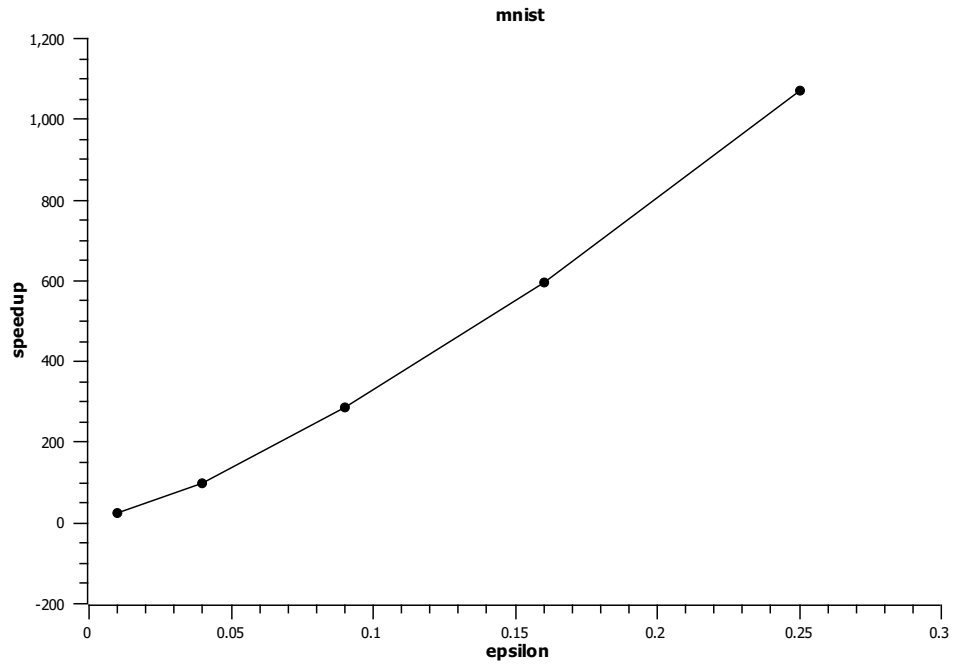


Figure 36: Speedup vs. error tolerance for mnist.

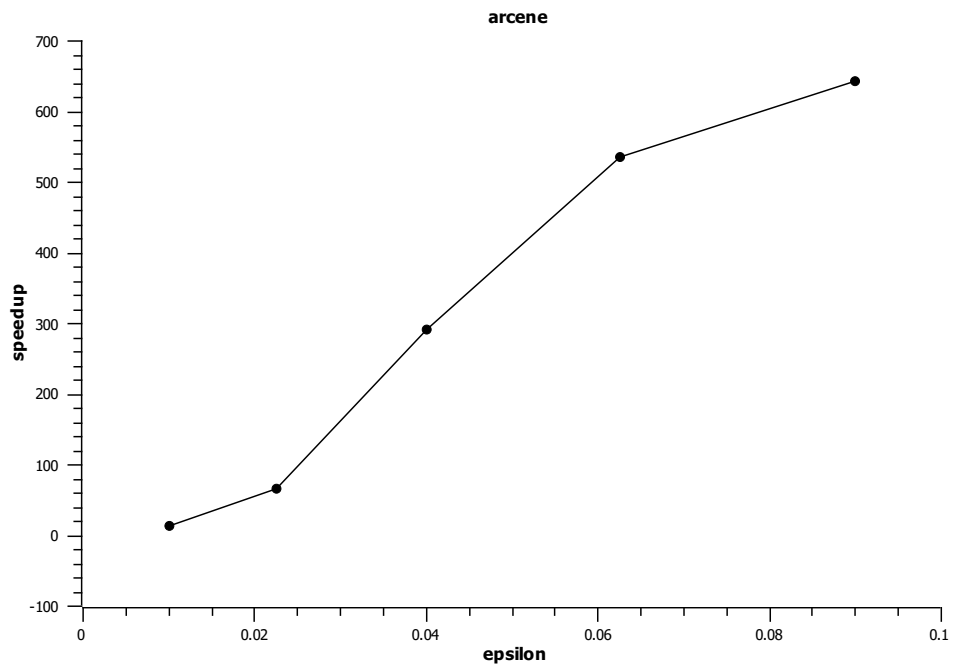
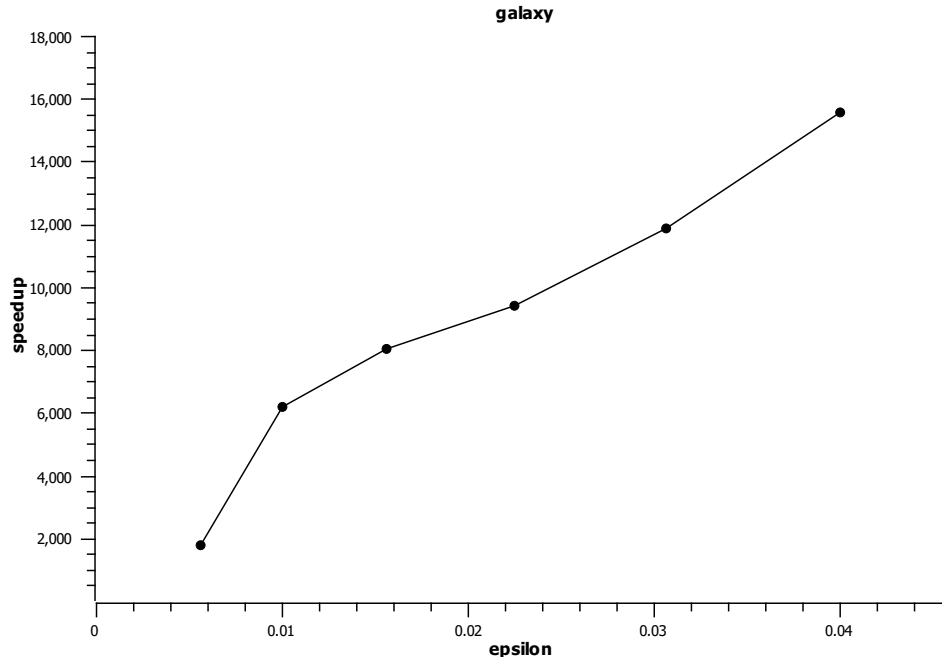


Figure 37: Speedup vs. error tolerance for arcene.



**Figure 38:** Speedup vs. error tolerance for galaxy.

### 5.5.3 Scalability on Large Matrices

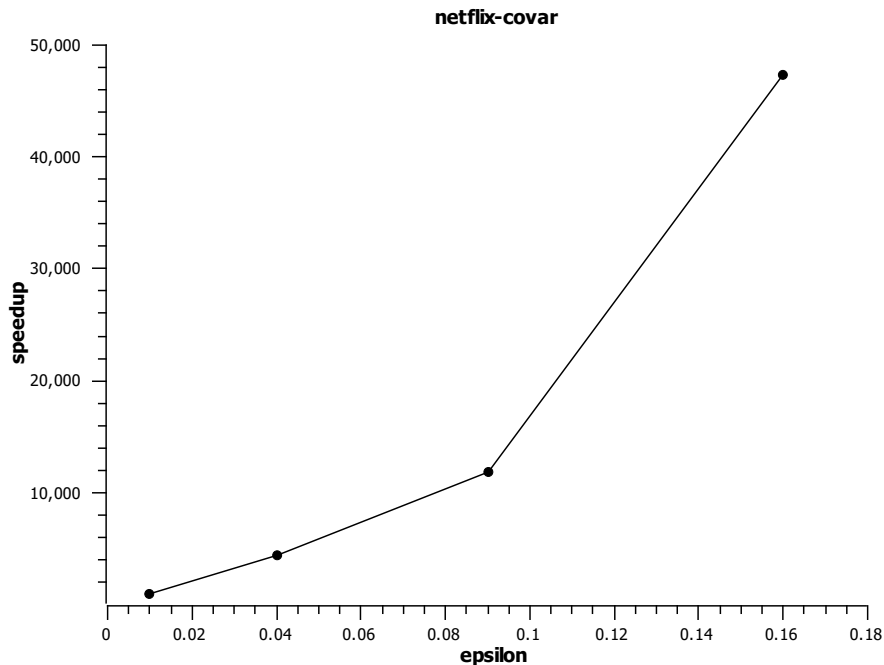
Our last set of experiments tests the scalability of QUIC-SVD on large matrices containing hundreds of millions to billions of entries, i.e., the size of our 16GB RAM. Table 9 describes these matrices. QUIC-SVD was run over a range of  $\epsilon$  values, with performance averaged over five runs as before. In the case of these large matrices, however, it was not practical for us to compute the exact SVD for runtime comparison. Instead, we ran exact SVD on subsets of the rows, then extrapolated the exact runtime based on its  $O(mn^2)$  growth order. In general this tends to underestimate the exact SVD time, so the speedups we give here can be interpreted as conservative estimates. Figures 39–41 show the speedup performance with respect to  $\epsilon$ .

As is apparent from the plots, speedup increases with  $\epsilon$  just as in the medium-scale case. The magnitudes of the speedups, however, are much larger, ranging from about 1,000 to 50,000 for netflix-covar, 2,000 to 14,500 for zeta, and 3,000 to 339,000 for dna-kernel. Note how the thin zeta matrix is significantly less accelerated than

**Table 9:** Descriptions of the three matrices used for large-scale experiments, ranging from 0.3 to 1.8 billion entries in size.

name	size	description
netflix-covar	$17,700 \times 17,700$	movie-movie covariance from Netflix prize dataset
zeta	$900,000 \times 2,000$	from ICML 2008 Pascal Large Scale Learning Challenge (PLSLC)
dna-kernel	$37,500 \times 37,500$	kernel matrix derived from subsampled PLSLC dna dataset

the two square matrices. This is again in accord with the  $O(\frac{n}{k})$  order of speedup. In absolute terms, we have reduction from 1.2 years to 2.6 hrs at 2.25% squared error for dna-kernel, from 16.5 days to 2 minutes at 1% squared error for zeta, and from 54 days to 1.4 hours at 1% squared error for netflix-covar. Again, we emphasize the qualitative leap in performance that makes QUIC-SVD a different creature, in terms of speed and scalability, than the exact SVD. Overall, the several-order-of-magnitude speedups and controlled error would seem to make QUIC-SVD an attractive option for any algorithm computing large-scale or speed-imperative SVDs.



**Figure 39:** Speedup vs. error tolerance for netflix-covar.

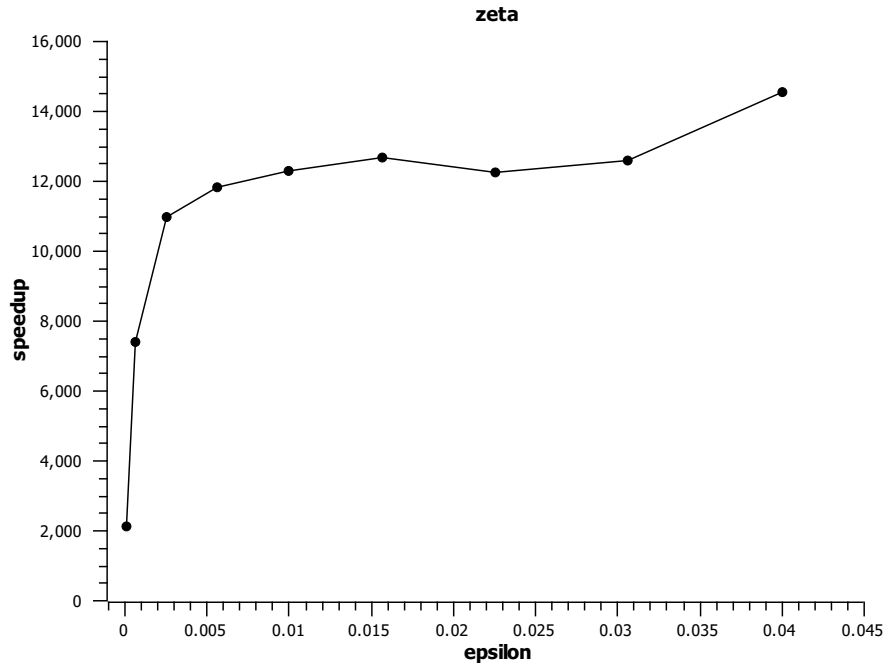


Figure 40: Speedup vs. error tolerance for zeta.

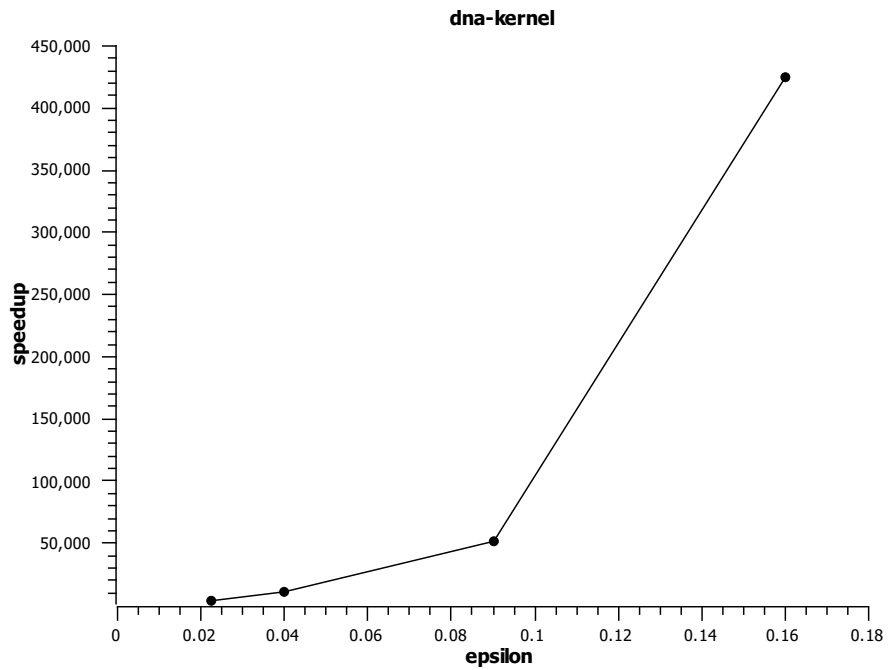


Figure 41: Speedup vs. error tolerance for dna-kernel.

## 5.6 Conclusion

We have presented a fast approximate SVD algorithm, QUIC-SVD, and demonstrated speedups as high as  $10^4$  and  $10^5$  on dataset matrices containing billions of entries. These speedups represent reductions in computation time from years to hours and from days to minutes. QUIC-SVD differs from previous related work in that it 1) addresses whole-matrix SVD approximation rather than low-rank matrix approximation, 2) uses tight, empirically-driven error control, and 3) employs a new, more efficient sampling procedure based on iterative cosine tree expansion. We have demonstrated that cosine tree sampling empirically appears to dominate previous sampling techniques in terms of subspace construction efficiency, with performance remarkably close to optimal in several instances. We have also shown that QUIC-SVD provides an automatic relative error guarantee with user-controlled error tolerance, which allows the user to choose the tradeoff between speed and accuracy according to the needs of particular SVD applications.

As a method combining spatial partitioning trees with sampling, QUIC-SVD is also clearly an instantiation of the Multi-Tree Monte Carlo methodology. It is a particularly important instance because it moves Multi-Tree Monte Carlo out of the realm of  $n$ -body-type problems and into the realm of linear algebra. This is suggestive of other potential linear algebraic applications of the MTMC methodology. It also suggest expanding MTMC into still other areas containing additional distinctive computational forms, such as the various branches of optimization. By extracting the commonalities among the various MTMC methods we have shown thus far, we can construct a general algorithmic framework that will form the foundation for thinking about how to apply MTMC to new computational forms. This framework is the substance of our final chapter.

## CHAPTER VI

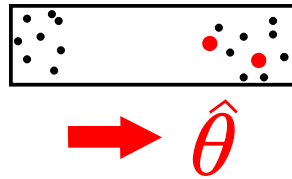
### THE MULTI-TREE MONTE CARLO FRAMEWORK

As we have seen, our approximation methods use several different patterns of interaction between trees and sampling. The fast KCDE training method of Chapter 3 injects Monte Carlo estimates into a dual-tree expansion, the recursive Monte Carlo algorithm of Chapter 4 uses trees in stratified sampling, and the QUIC-SVD of Chapter 5 uses a tree to guide sampling, which in turn guides the expansion of the tree. Thus, three modalities of Multi-Tree Monte Carlo have emerged: Monte Carlo in trees, trees in Monte Carlo, and reciprocally interacting trees and Monte Carlo. In this chapter we seek to unify these modalities into a general framework to help us distill the essence and explore the scope of the methodology. We give a framework meta-algorithm that formalizes and generalizes the patterns used by the algorithms we have developed. This meta-algorithm does not directly derive new MTMC algorithms, but rather provides a seed point to which we can return for guidance as we derive new approximation algorithms. Though the individual algorithms presented in the foregoing chapters have shown large efficiency gains several orders of magnitude beyond the prior state of the art, the idea of the *methodology* may ultimately be a greater contribution as the “gift that keeps on giving.”

What are the essential characteristics of Multi-Tree Monte Carlo? Given an object  $\theta$  to be estimated with error tolerance  $\epsilon$ , a MTMC algorithm features iterative buildup of an approximation  $\hat{\theta}$ , frequent error checking to enable the soonest possible termination, and the interleaving of trees and sampling to guide the buildup and error estimation. We formalize this with the meta-algorithm schematic shown in Figure 6.

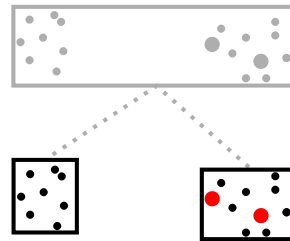
MULTI-TREE MONTE CARLO

1. Obtain an initial estimate  $\hat{\theta}$  using an initial partitioning and sampling.

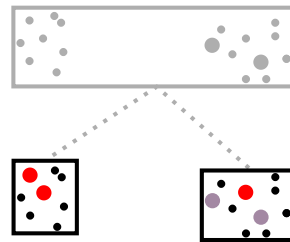


2. **while**  $err(\hat{\theta}) > \epsilon$

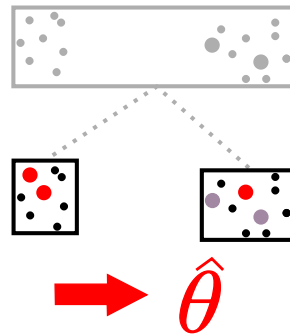
- (a) (optional) Refine the partitioning in light of  $\hat{\theta}$ ,  $err(\hat{\theta})$ , and samples drawn.



- (b) Draw additional samples guided by the partitioning,  $\hat{\theta}$ , and  $err(\hat{\theta})$ .



- (c) Refine or recalculate  $\hat{\theta}$ .



- (d) Recalculate or reestimate  $err(\hat{\theta})$ .

**Figure 42:** Multi-Tree Monte Carlo meta-algorithm schematic.

To illustrate how the MTMC meta-algorithm leads to concrete MTMC methods for specific problems, consider how the methods from Chapters 3–5 fit into the framework. First we examine MCLL, the fast KCDE training method of Chapter 3, which represents the pattern of *Monte Carlo in trees* (see Algorithm 3.3). Its initial partitioning consists of dual *kd*-tree roots, and its initial estimate of the log-likelihood is based on pairs of samples between those nodes. This is step 1 of the meta-algorithm. If the estimate does not satisfy the error bound sufficient conditions we derived, we iteratively split a frontier node (step 2(a)), resample from pairs of frontier nodes to get new estimates (steps 2(b)-(c)), and re-check the error bound conditions (step 2(d)). This is a clear instantiation of the meta-algorithm

Next, consider RSMC, our recursive Monte Carlo algorithm for nested-summative forms such as kernel estimators (Algorithm 4.4). RSMC represents the pattern of *trees in Monte Carlo*. It begins by expanding a tree until its frontier contains a user-specified number of nodes (strata). This is framework step one. The initial error estimate is effectively infinite, and the algorithm proceeds by iteratively sampling from the tree strata (step 2(b)), folding the new samples into the stratified sample mean (step 2(c)), and estimating error based on the stratified sample variance (step 2(d)). Thus, RSMC also fits squarely into the framework. Note that step 2(a) is skipped in this case, since the tree is never refined beyond its initial expansion.

Lastly, QUIC-SVD (Algorithm 5.3) also unifies with the meta-algorithm, and represents the mode of *reciprocal trees and Monte Carlo*. It begins with the default unpartitioned dataset and chooses its mean as a sample point. This sample point is used to construct the initial subspace estimate, which implicitly represents the approximate SVD. These constitute step 1 of the framework. Then, iteratively, the algorithm splits the partition with highest error based on a sampled split point (step 2(a)), takes the right child’s mean as the next sample point for the subspace (step 2(b)), expands the subspace to span the new sample point (step 2(c)), and updates

the error using projection onto the new subspace dimension (step 2(d)).

Thus we see that while each algorithm is unique in its particulars, they all flow within the framework described by the MTMC meta-algorithm. It should be clear that the methodology represented by the meta-algorithm can be used to derive a host of additional approximation methods. Certainly methods with forms similar to those we have already addressed are likely to be amenable to similar acceleration techniques. For instance, many methods in machine learning require costly kernel matrix inversions. We could accelerate such inversions and even avoid explicit construction of the kernel matrix using a kernelized cosine tree and techniques similar to the QUIC-SVD. Kernel machines such as SVMs have both a summative and linear algebraic side to them, which suggests a connection to our summative-form and SVD approximations. Graphical model inference is related to solving linear systems, which can be done by the SVD. On the other hand, we also have computational forms entirely distinct from summative and linear algebraic forms that may also yield to MTMC approximation. Optimization, for instance, is a particularly attractive area because of its ubiquity, and presents many enticing targets such as linear, quadratic, and semi-definite programming.

Obviously there are many avenues to explore, and it is especially fruitful to concentrate on core computational engines like the SVD that are widely used as components in higher-level methods. Acceleration of such fundamental computations can produce widespread uplift and impact. It is our hope that elaborating the Multi-Tree Monte Carlo methodology as we have done may help to quicken the pace and guide the course of these developments.

## CHAPTER VII

### CONCLUSION

Multi-Tree Monte Carlo is a new approach to error-controlled algorithmic acceleration of machine learning and related methods. It combines the spatial partitioning aspect of classical multi-tree methods with Monte Carlo and other sampling techniques. In doing so, an essential compromise is made: high computation speed at the price of losing determinism. In the large-data, high-responsiveness applications that are quickly becoming the norm, this is arguably very often the right sacrifice to make. Using the MTMC methodology, we have obtained the following fast approximation methods:

1. Fast training for kernel conditional density estimation by injecting Monte Carlo into state-of-the-art dual-tree methods. Speedups as high as  $10^5$  have been shown on datasets of up to 1 million points.
2. Fast training for general kernel estimators and other nested-summative forms by injecting multiple trees into Monte Carlo. Speedups as high as  $10^6$  have been shown on tens of millions of points.
3. Fast singular value decomposition using a new form of sampling tree called the cosine tree. Speedups as high as  $10^5$  have been shown on dataset matrices containing billions of entries.

These results combine to support our central thesis, which is that the Multi-Tree Monte Carlo methodology can be used to produce approximate machine learning methods that are: 1) highly accelerated, 2) error-controlled, and 3) scalable to large datasets.

As discussed in the Introduction, this thesis consists of three subclaims, and establishing it requires us to establish each subclaim as it pertains to the methods we have developed (see Figure 1). High acceleration has certainly been shown in the empirical speedups demonstrated by each method across a broad array of datasets. Error control has been shown for each method, both theoretically and with empirical verification. Lastly, scalability has been evidenced by highly efficient performance as we have applied each method to large-scale data. Thus, the claims of the thesis have been validated by the body of work presented herein.

### ***7.1 Future Extensions***

The continued development of the Multi-Tree Monte Carlo methodology should include at least two key thrusts: 1) expanding the set of fast approximation algorithms derived using MTMC techniques, and 2) refining, generalizing, and improving the reusability of the framework.

On the first point, there are algorithms that can fairly obviously be accelerated by MTMC in the near term, as well as algorithmic areas that would require substantially more research to determine whether they can be helped by MTMC. In the near-term category, we have  $n$ -body-type problems related to the kernel estimator training methods described in Chapters 3-4. These include problems such as evaluating kernel estimators on a per-point basis over large datasets or over many bandwidths at once, nearest-neighbor problems, range searches, and so on. For the RSMC algorithm of Chapter 4 in particular, we have not yet explored the application to non-kernel-estimator problems, such as Gaussian mixture models, support vector machines, and stochastic gradient descent, all of which involve nested-summative forms of the sort we have addressed. Also in the near term are extensions of the QUIC-SVD. For instance, we have already mentioned the possibility of kernelizing QUIC-SVD in order to accelerate methods that require costly inversion or decomposition of a kernel

matrix (kernel PCA, Gaussian process regression, kernel ridge regression, etc.). A “kernel tree” or “kernel cosine tree” might even be able to avoid explicit construction of the  $n \times n$  kernel matrix. Various enhancements to the cosine tree sampling method may also be possible, e.g., by combining it with ideas from Arnoldi iteration. It may even be possible to combine QUIC-SVD and RSMC directly in an approximation method for local linear regression.

In the longer term are applications to areas such as optimization, different linear algebra problems, and core machine learning methods we have not yet considered. Some of these are convergent; for instance, some types of inference and learning in graphical models appear to reduce to solving linear systems. In this case, a core machine learning computation would be accelerated through a fast linear algebra solver ((QUIC-SVD can already address this problem). Similarly, the inner loop of Newton-style optimization methods requires inversion of the Hessian matrix, which could be approximated directly by QUIC-SVD, or perhaps done even more efficiently by creating tree structures that avoid explicit construction of the Hessian. Such an advance could have large impact due to the wide use of Newton-type optimization techniques. Additionally, we may consider other linear algebraic decompositions that find important uses in machine learning, such as the QR decomposition, the generalized SVD, and so on. Semi-definite programming, which has gained recent popularity in machine learning, may be amenable to a MTMC approach. MTMC acceleration of support vector machine training and evaluation could have high impact, and seems plausible given the linear-algebraic and summative forms of those computations. The possibilities are extensive, and it seems wise to focus on the core-most operations such that scalable MTMC approximations will generate the most widespread uplift.

In the course of expanding Multi-Tree Monte Carlo to cover a wider range of algorithms, the methodology will likely be significantly refined. For instance, the error criteria we have used in the foregoing chapters may not be right for future extensions.

For linear algebraic applications, perhaps error under the Frobenius norm is not well suited for some problems – perhaps error bounds based on the  $L_1$ ,  $L_\infty$ , or other norms should be made available. It may be that incorporating supervision (class labels, regression targets) will make for better error criteria, or even faster runtimes. It seems likely that the set of spatial partitioning schemes will have to grow to accommodate new applications. We have already mentioned the idea of kernelizing the cosine tree or developing a Hessian-oriented tree. Many other schemes are possible, including hybrid Monte Carlo/deterministic approaches. Likewise, the set of sampling techniques and sample-based error bounds used in MTMC approximations is currently small, but could be augmented by things like quasi-Monte Carlo sampling, non-asymptotic confidence bounds expressed purely in sample statistics, data-driven versions of standard variance reduction techniques (control variates, antithetic variables, etc.; see [76, 61]), and so on. In all of this, however, the core idea of “divide and sample” remains the guiding principle, and the high-level meta-algorithm described in Chapter 6 seems likely to remain fairly stable.

## ***7.2 Implications for Machine Learning Applications***

Our motivation in the development of Multi-Tree Monte Carlo has been to enable machine learning to handle the massive data demands of modern applications both present and future. Our goal has been the development of scalable, approximate, error-controlled versions of machine learning algorithms, with a focus on accelerating core computational bottlenecks in order to generate widespread benefit to the many algorithms that depend on those bottlenecks. We have provided both a general methodology for producing scalable algorithms as well as specific fast approximations for the SVD, kernel density estimation, kernel regression, kernel conditional density estimation, and an entire class of nested-summative computational forms.

The SVD and kernel density estimation in particular are very widely used both

directly and as components in higher-level machine learning algorithms. Methods that use or could use the SVD include PCA, kernel PCA, general nonlinear manifold learning (by virtue of reduction to kernel PCA), many clustering algorithms, latent semantic indexing, SVD-based approaches to collaborative filtering, metric-learning approaches based on eigenvalue problems, approaches to DNA microarray reconstruction, and so on [128, 54, 57, 34, 107, 27, 10, 62, 43, 18, 143, 50]. Kernel density estimation solves a fundamental statistical problem and is therefore often used directly, but other methods that employ it include versions of particle filtering, clustering methods such as the mean-shift or typical-cut algorithms, nonparametric Bayes classification, feature extraction methods such as tree-dependent component analysis (a generalization of ICA), computer vision applications such as segmentation and tracking, and so on [22, 139, 46, 118, 7, 45].

Thus, by choosing such core computations as the targets for our fast approximations, the few methods we have provided here are like leaven in the loaf, as they can provide an uplift to scalability for the many more sophisticated methods that make use of them. This, in turn, will enable the use of those more sophisticated methods on large-scale datasets, allowing them to keep up with the demanding pace of cutting-edge applications.

There remains, however, much to be explored in terms of the effect that approximation will have on the *statistical* performance of higher-level algorithms. While in this work we have endeavored to provide sensible forms of error control, it may be that the type or degree of error we have examined here will not be appropriate for all higher-level algorithms. For instance, suppose we use QUIC-SVD to compute PCA for feature extraction or visualization. One might imagine that a fairly substantial squared Frobenius-norm error on the order of 10% would be acceptable. When using SVD to solve a linear system, however, it may be that we would need much smaller error, or that the bound should be on the  $L_\infty$  norm (maximum per-entry error).

With respect to kernel estimators, it is frequently the case that objective functions over bandwidths have wide, flat global optima, such that a large range of bandwidths are essentially equivalent to the optimal [140]. In such cases, very high relative error can be allowed on approximate bandwidth scores, since it is primarily the order of magnitude that matters. For other nested-summative computations, however, it may be that much tighter relative error bounds will be required, or that a bound on some auxiliary quantity such as divergence in training error will prove more useful.

Clearly these issues will have to be explored for each higher-level algorithm or class of higher-level algorithms for which we wish to use our fast approximations. The results presented in Chapters 3–4 show that the level of speedup rises with the level of error tolerance. Algorithms that can use our approximations at higher error levels will therefore get the most benefit, while algorithms needing tighter error control will see diminishing returns the less error they allow. In our experiments, we have focused on relative error bounds in the range of 10%–40% for kernel method training and 0.1%–25% for the SVD. These error levels have yielded speedups of several orders of magnitude and more. Beneath these ranges, speedups are likely to decrease to the level of  $10 - 10^2$  for datasets similar in size to those we have addressed. Above these ranges there may be additional speedup, though in some instances speedup reaches a plateau near the top of the range. It seems fair to say that, in our experience thus far, these error ranges represent the “sweet spot” of the speed/error tradeoff for our MTMC methods. For higher-level algorithms that can use our fast approximations with error levels in or near these ranges, MTMC should enable applications at scales many orders of magnitude higher than previously feasible.

### ***7.3 Summary***

The move to Multi-Tree Monte Carlo approximation is a little like going from a steam engine to a rocket. There is great power that, if properly harnessed, can enable

entirely new and unconceived-of journeys; but if proper care is not taken, control can easily be lost. Also, while a rocket is orders of magnitude more powerful than a steam engine, it is not the right thing for every situation. Hence, one must be able to both wield control and appreciate appropriateness.

Our methods have represented several different modes of Multi-Tree Monte Carlo: Monte Carlo in trees, trees in Monte Carlo, and reciprocal interaction between trees and Monte Carlo. These modes are unified in the MTMC meta-algorithm described in Chapter 6, which formalizes the general MTMC methodology and serves as scaffolding for thinking about how to develop MTMC approximations for methods beyond those considered here. Taken together, we hope the various pieces of this work have established a valuable approach to deriving fast and accurate approximate versions of algorithms from machine learning, data mining, and other fields with related computational forms. This will fill an important need as algorithms grow ever more sophisticated and datasets ever larger, and should help machine learning and related methods to successfully meet the ever more extreme demands that will be placed on them by applications both now and into the foreseeable future.

## REFERENCES

- [1] ACHLIOPTAS, D., MCSHERRY, F., and SCHOLKOPF, B., “Sampling Techniques for Kernel Methods,” in *Advances in Neural Information Processing Systems (NIPS) 17*, 2002.
- [2] ACHLIOPTAS, D. and MCSHERRY, F., “Fast computation of low rank matrix approximations,” in *Proceedings of the 33rd Annual ACM Symposium on Theory of Computing*, 2001.
- [3] ANDERSON, C., “The End of Theory: The Data Deluge Makes the Scientific Method Obsolete,” *Wired*, June 2008.
- [4] ANDRIEU, C., DOUCET, A., and PUNSKAYA, E., “Sequential Monte Carlo Methods for Optimal Filtering,” in *Sequential Monte Carlo Methods in Practice*, Springer-Verlag, 2001.
- [5] ANDRIEU, C., DE FREITAS, N., DOUCET, A., and JORDAN, M. I., “An Introduction to MCMC for Machine Learning,” *Machine Learning*, vol. 50, pp. 5–43, 2003.
- [6] ATIYA, A. F. and PARLOS, A. G., “New results on recurrent network training: unifying the algorithms and accelerating convergence,” *IEEE Transactions on Neural Networks*, vol. 11, no. 3, pp. 697–709, 2000.
- [7] BACH, F. R. and JORDAN, M. I., “Tree-dependent Component Analysis,” in *Uncertainty in Artificial Intelligence (UAI)*, 2002.
- [8] BASHTANNYK, D. M. and HYNDMAN, R. J., “Bandwidth selection for kernel conditional density estimation,” *Computational Statistics & Data Analysis*, vol. 36, pp. 279–298, May 2001.
- [9] BAUM, L. E., PETRIE, T., SOULES, G., and WEISS, N., “A Maximization Technique Occurring in the Statistical Analysis of Probabilistic Functions of Markov Chains,” *Annals of Mathematical Statistics*, vol. 41, pp. 164–171, 1970.
- [10] BERRY, M. W., DUMAIS, S. T., and O’BRIEN, G. W., “Using linear algebra for intelligent information retrieval,” *SIAM Review*, vol. 37, no. 4, pp. 573–595, 1995.
- [11] BLUM, A. and RIVEST, R. L., “Training a 3-Node Neural Network is NP-Complete (Extended Abstract),” in *1988 Workshop on Computational Learning Theory*, pp. 9–18, Morgan Kaufmann, 1988.

- [12] BORDES, A., USUNIER, N., and BOTTOU, L., “Sequence Labelling SVMs Trained in One Pass,” in *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD)*, 2008.
- [13] BOTTOU, L., “Stochastic Gradient Learning in Neural Networks,” in *Proceedings of Neuro-Nimes 91*, (Nimes, France), EC2, 1991.
- [14] BOTTOU, L., “Online algorithms and stochastic approximations,” in *Online Learning and Neural Networks* (SAAD, D., ed.), Cambridge, UK: Cambridge University Press, 1998.
- [15] BOTTOU, L. and BOUSQUET, O., “The tradeoffs of large scale learning,” in *Advances in Neural Information Processing Systems (NIPS) 20*, Cambridge, MA: MIT Press, 2008.
- [16] BOTTOU, L. and LECUN, Y., “Large scale online learning,” in *Advances in Neural Information Processing Systems (NIPS) 16*, Cambridge, MA: MIT Press, 2004.
- [17] BOYER, G. F., RIEGEL, R. N., and GRAY, A. G., “A Parallel  $N$ -Body Data Mining Framework,” in *NIPS Workshop on Efficient Machine Learning*, 2007.
- [18] BRAND, M., “Fast Online SVD Revisions for Lightweight Recommender Systems,” in *SIAM International Conference on Data Mining*, 2003.
- [19] BREIMAN, L., “Bagging Predictors,” *Machine Learning*, vol. 24, pp. 123–140, August 1996.
- [20] CELEUX, G. and DIEBOLT, J., “The SEM Algorithm: A Probabilistic Teacher Algorithm Derived from the EM Algorithm for the Mixture Problem,” *Computational Statistics Quarterly*, vol. 2, pp. 73–82, 1985.
- [21] CELEUX, G. and DIEBOLT, J., “A Stochastic Approximation Type EM Algorithm for the Mixture Problem,” *Stochastics and Stochastics Reports*, vol. 41, pp. 127–146, 1992.
- [22] CHENG, C., ANSARI, R., and KHOKHAR, A., “Multiple object tracking with kernel particle filter,” in *Computer Vision and Pattern Recognition (CVPR)*, 2005.
- [23] CHICKERING, D. M., “Learning Bayesian networks is NP-complete,” in *Learning from Data: Artificial Intelligence and Statistics V*, pp. 121–130, Springer-Verlag, 1996.
- [24] CHICKERING, D. M., HECKERMAN, D., and MEEK, C., “Large-Sample Learning of Bayesian Networks is NP-Hard,” *Journal of Machine Learning Research*, vol. 5, pp. 1287–1330, October 2004.

- [25] DE FREITAS, N., NIRANJAN, M., GEE, A. H., and DOUCET, A., “Sequential Monte Carlo methods to train neural network models,” *Neural Computation*, vol. 12, no. 4, pp. 955–993, 2000.
- [26] DE FREITAS, N., WANG, Y., MAHDAVIANI, M., and LANG, D., “Fast Krylov Methods for N-Body Learning,” in *Advances in Neural Information Processing Systems (NIPS) 18*, 2005.
- [27] DEERWESTER, S. C., DUMAIS, S. T., LANDAUER, T. K., FURNAS, G. W., and HARSHMAN, R. A., “Indexing by Latent Semantic Analysis,” *Journal of the American Society of Information Science*, vol. 41, no. 6, pp. 391–407, 1990.
- [28] DELLAERT, F., KWATRA, V., and OH, S. M., “Mixture Trees for Modeling and Fast Conditional Sampling with Applications in Vision and Graphics,” in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2005.
- [29] DELLAERT, F., SEITZ, S. M., THORPE, C. E., and THRUN, S., “EM, MCMC, and Chain Flipping for Structure from Motion with Unknown Correspondences,” *Machine Learning*, vol. 50, pp. 45–71, 2003.
- [30] DEMPSTER, A. P., LAIRD, N. M., and RUBIN, D. B., “Maximum Likelihood from Incomplete Data via the EM Algorithm,” *Journal of the Royal Statistical Society Series B*, vol. 39, pp. 1–38, 1997.
- [31] DESHPANDE, A. and VEMPALA, S., “Adaptive Sampling and Fast Low-Rank Matrix Approximation,” in *10th International Workshop on Randomization and Computation (RANDOM06)*, 2006.
- [32] DESHPANDE, A., RADEMACHER, L., VEMPALA, S., and WANG, G., “Matrix Approximation and Projective Clustering via Volume Sampling,” in *Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2006.
- [33] DESHPANDE, A., RADEMACHER, L., VEMPALA, S., and WANG, G., “Matrix Approximation and Projective Clustering via Volume Sampling,” *Theory of Computing*, vol. 2, pp. 225–247, 2006.
- [34] DING, C. and HE, X., “K-means Clustering via Principal Component Analysis,” in *International Conference on Machine Learning (ICML)*, 2004.
- [35] DOMINGOS, P. and HULTEN, G., “Mining high-speed data streams,” in *International Conference on Knowledge Discovery and Data Mining (KDD)*, 2000.
- [36] DOMINGOS, P. and HULTEN, G., “A general method for scaling up machine learning algorithms and its application to clustering,” in *International Conference on Machine Learning (ICML)*, pp. 106–113, Morgan Kaufmann, 2001.

- [37] DOUCET, A., GODSILL, S., and ANDRIEU, C., “On sequential Monte Carlo sampling methods for Bayesian filtering,” *Statistics and Computing*, vol. 10, pp. 197–208, 2000.
- [38] DRINEAS, P., DRINEA, E., and HUGGINS, P. S., “An Experimental Evaluation of a Monte-Carlo Algorithm for Singular Value Decomposition,” *Lecture Notes in Computer Science*, vol. 2563, pp. 279–296, 2003.
- [39] DRINEAS, P., KANNAN, R., and MAHONEY, M. W., “Fast Monte Carlo Algorithms for Matrices II: Computing a Low-Rank Approximation to a Matrix,” *SIAM Journal on Computing*, vol. 36, no. 1, pp. 158–183, 2006.
- [40] DRINEAS, P., FRIEZE, A., KANNAN, R., VEMPALA, S., and VINAY, V., “Clustering in Large Graphs and Matrices,” in *SODA: ACM-SIAM Symposium on Discrete Algorithms*, 1999.
- [41] DRINEAS, P. and KANNAN, R., “Fast Monte-Carlo Algorithms for Approximate Matrix Multiplication,” in *Proceedings of the 42nd IEEE Symposium on Foundations of Computer Science (FOCS)*, 2001.
- [42] DRINEAS, P., KANNAN, R., and MAHONEY, M. W., “Fast Monte Carlo Algorithms for Matrices I: Approximating Matrix Multiplication,” *SIAM Journal on Computing*, vol. 36, no. 1, pp. 132–157, 2006.
- [43] DRINEAS, P., KERENIDIS, I., and RAGHAVAN, P., “Competitive Recommendation Systems,” in *Proceedings of the 34th ACM Symposium on Theory of Computing (STOC)*, 2002.
- [44] EFRON, B., “Bootstrap Methods: Another Look at the Jackknife,” *The Annals of Statistics*, vol. 7, no. 1, pp. 1–26, 1979.
- [45] ELGAMMAL, A., DURAISWAMI, R., and DAVIS, L. S., “Efficient Computation of Kernel Density Estimation using Fast Gauss Transform with Applications for Segmentation and Tracking,” in *ICCV Workshop on Statistical and Computational Theories of Vision*, 2001.
- [46] ERDOGMUS, D., CARREIRA-PERPINAN, M. A., and OZERTEM, U., “Kernel Density Estimation, Affinity-Based Clustering, and Typical Cuts,” in *International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2006.
- [47] ESTIVILL-CASTRO, V. and MCKENZIE, B. S., “Hierarchical Monte-Carlo Localization Balances Precision and Speed,” in *Australasian Conference on Robotics and Automation*, 2004.
- [48] FAN, J., YAO, Q., and TONG, H., “Estimation of conditional densities and sensitivity measures in nonlinear dynamical systems,” *Biometrika*, vol. 83, no. 1, pp. 189–206, 1996.

- [49] FAN, J. and YIM, T. H., “A crossvalidation method for estimating conditional densities,” *Biometrika*, vol. 91, no. 4, pp. 819–834, 2004.
- [50] FRIEDLAND, S., NIKNEJAD, A., and CHIHARA, L., “A Simultaneous Reconstruction of Missing Data in DNA Microarrays,” *Linear Algebra and its Applications*, vol. 416, no. 1, pp. 2–28, 2006.
- [51] FRIEDLAND, S., NIKNEJAD, A., KAVEH, M., and ZARE, H., “Fast Monte-Carlo Low Rank Approximations for Matrices,” in *Proceedings of the 2006 IEEE/SMC International Conference on System of Systems Engineering*, 2006.
- [52] FRIEZE, A., KANNAN, R., and VEMPALA, S., “Fast Monte-Carlo Algorithms for Finding Low-Rank Approximations,” *Journal of the ACM (JACM)*, vol. 51, no. 6, pp. 1025–1041, 2004.
- [53] FRIEZE, A. M., KANNAN, R., and VEMPALA, S., “Fast Monte-Carlo Algorithms for Finding Low-Rank Approximations,” in *IEEE Symposium on Foundations of Computer Science*, pp. 370–378, 1998.
- [54] FRIEZE, P. D. A., KANNAN, R., VEMPALA, S., and VINAY, V., “Clustering Large Graphs via the Singular Value Decomposition,” *Machine Learning*, vol. 56, pp. 9–33, 2004.
- [55] FURNKRANZ, J., “Integrative Windowing,” *Journal of Artificial Intelligence Research*, vol. 8, pp. 129–164, 1998.
- [56] FURNKRANZ, J., PETRAK, J., BRAZDIL, P., and SOARES, C., “On the use of fast subsampling estimates for algorithm recommendation,” tech. rep., sterreichisches Forschungsinstitut fr Artificial Intelligence, 2002.
- [57] GE, X., PARISE, S., and SMYTH, P., “Clustering Markov States into Equivalence Classes using SVD and Heuristic Search Algorithms,” in *International Workshop on Artificial Intelligence and Statistics (AISTATS)*, 2003.
- [58] GHAHRAMANI, Z., “Factorial Learning and the EM Algorithm,” in *Advances in Neural Information Processing Systems (NIPS) 7*, 1995.
- [59] GHAHRAMANI, Z. and JORDAN, M., “Factorial Hidden Markov Models,” tech. rep., MIT, 1995.
- [60] GILKS, W. R., THOMAS, A., and SPIEGELHALTER, D. J., “A language and program for complex Bayesian modeling,” *The Statistician*, vol. 43, pp. 169–178, 1994.
- [61] GLASSERMAN, P., *Monte Carlo Methods in Financial Engineering*. Springer-Verlag, 2004.
- [62] GOLDBERG, K., ROEDER, T., GUPTA, D., and PERKINS, C., “Eigentaste: A Constant Time Collaborative Filtering Algorithm,” *Information Retrieval*, vol. 4, no. 2, pp. 133–151, 2001.

- [63] GONZALEZ, T., “Clustering to minimize the maximum intercluster distance,” *Theoretical Computer Science*, vol. 38, pp. 293–306, 1985.
- [64] GOOIJER, J. G. D. and ZEROM, D., “On Conditional Density Estimation,” *Statistica Neerlandica*, vol. 57, pp. 159–176, May 2003.
- [65] GRAY, A. G., “Fast Kernel Matrix-Vector Multiplication with Application to Gaussian Process Regression,” tech. rep., Carnegie Mellon Computer Science Department, 2004.
- [66] GRAY, A. G. and MOORE, A. W., “Nonparametric Density Estimation: Toward Computational Tractability,” in *SIAM International Conference on Data Mining (SDM)*, 2003.
- [67] GRAY, A. G. and MOORE, A. W., “Very Fast Multivariate Kernel Density Estimation via Computational Geometry,” in *Proceedings of the ASA Joint Statistical Meeting*, 2003.
- [68] GRAY, A. G. and MOORE, A. W., “Fast Computation of the Pair Correlation and  $n$ -Point Correlation Functions,” in *Conference on Computational Physics*, 2004.
- [69] GRAY, A. G., MOORE, A. W., NICHOL, R. C., CONNOLLY, A. J., GENOVESE, C., and WASSERMAN, L., “Multi-Tree Methods for Statistics on Very Large Datasets in Astronomy,” in *Astronomical Data Analysis Software and Systems (ADASS) XIII* (OCHSENBEIN, F., ALLEN, M. G., and EGRET, D., eds.), 2004.
- [70] GRAY, A. and RIEGEL, R., “Large-Scale Kernel Discriminant Analysis with Application to Quasar Discovery,” in *CompStat*, 2006.
- [71] GRAY, A. G., *Bringing Tractability to Generalized  $N$ -body Problems in Statistical and Scientific Computations*. PhD thesis, Carnegie Mellon University Computer Science Department, 2003.
- [72] GRAY, A. G. and MOORE, A. W., “ $N$ -Body Problems in Statistical Learning,” in *Advances in Neural Information Processing Systems (NIPS) 13*, 2000.
- [73] GRAY, A. G. and MOORE, A. W., “Rapid Evaluation of Multiple Density Models,” in *Workshop on Artificial Intelligence and Statistics (AISTATS)*, 2003.
- [74] GREENGARD, L. and ROKHLIN, V., “A Fast Algorithm for Particle Simulations,” *Journal of Computational Physics*, vol. 73, 1987.
- [75] HAMALAINEN, P., AILA, T., TAKALA, T., and ALANDER, J., “Mutated  $Kd$ -tree Importance Sampling,” in *Scandinavian Conference on Artificial Intelligence (SCAI)*, 2006.

- [76] HAMMERSLEY, J. M. and HANDSCOMB, D. C., *Monte Carlo Methods*. Methuen, 1964.
- [77] HANSEN, B. E., “Nonparametric Conditional Density Estimation.” Unpublished manuscript, November 2004.
- [78] HAZARD, J., PALLATTO, J., and BURT, J., “Large Hadron Collider Smashes Particles and Crunches Numbers on a Massive Scale.” *eweek.com*, September 2008.
- [79] HOLMES, C. C. and MALLICK, B. K., “Bayesian Radial Basis Functions of Variable Dimension,” *Neural Computation*, vol. 10, no. 5, pp. 1217–1233, 1998.
- [80] HOLMES, M. P., GRAY, A. G., and ISBELL JR., C. L., “Fast Nonparametric Conditional Density Estimation,” in *Uncertainty in Artificial Intelligence (UAI)*, 2007.
- [81] HOLMES, M. P., GRAY, A. G., and ISBELL JR., C. L., “Ultrafast Monte Carlo for Kernel Estimators and Generalized Statistical Summations,” in *Advances in Neural Information Processing Systems (NIPS) 21*, 2008.
- [82] HOLMES, M. P., GRAY, A. G., and ISBELL JR., C. L., “QUIC-SVD: Fast SVD Using Cosine Trees,” in *Advances in Neural Information Processing Systems (NIPS) 22*, 2009.
- [83] HYAFIL, L. and RIVEST, R. L., “Constructing Optimal Binary Decision Trees is NP-Complete,” *Information Processing Letters*, vol. 5, no. 1, pp. 15–17, 1976.
- [84] HYNDMAN, R. J., BASHTANNYK, D. M., and GRUNWALD, G. K., “Estimating and Visualizing Conditional Densities,” *Journal of Computation and Graphical Statistics*, vol. 5, pp. 315–336, December 1996.
- [85] INDYK, P., “Algorithms for Massive Datasets, MIT Course 6.897 Lecture 6,” 2002.
- [86] ISARD, M. and BLAKE, A., “Contour tracking by stochastic propagation of conditional density,” in *European Conference on Computer Vision*, 1996.
- [87] JENSEN, C. S., KONG, A., and KJAERULF, U., “Blocking-Gibbs Sampling in Very Large Probabilistic Expert Systems,” *International Journal of Human-Computer Studies*, vol. 42, pp. 647–666, 1995.
- [88] JOHN, G. H. and LANGLEY, P., “Static versus dynamic sampling for data mining,” in *International Conference on Knowledge Discovery and Data Mining (KDD)*, pp. 367–370, AAAI Press, 1996.
- [89] JUDD, J. S., *Neural Network Design and the Complexity of Learning*. MIT Press, 1990.

- [90] KAJIYA, J. T., “The rendering equation,” in *Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)*, 1986.
- [91] KEARNS, M. and SINGH, S., “Near-optimal reinforcement learning in polynomial time,” in *Machine Learning*, pp. 260–268, Morgan Kaufmann, 1998.
- [92] KLAAS, M., BRIERS, M., DE FREITAS, N., and DOUCET, A., “Fast Particle Smoothing: If I Had a Million Particles,” in *International Conference on Machine Learning (ICML)*, 2006.
- [93] KLAAS, M., DE FREITAS, N., and DOUCET, A., “Toward Practical  $N^2$  Monte Carlo: The Marginal Particle Filter,” in *Uncertainty in Artificial Intelligence (UAI)*, 2005.
- [94] KLAAS, M., LANG, D., and DE FREITAS, N., “Fast Maximum a Posteriori Inference in Monte Carlo State Spaces,” in *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2005.
- [95] LANG, D., KLAAS, M., and DE FREITAS, N., “Empirical Testing of Fast Kernel Density Estimation Algorithms,” Tech. Rep. TR-2005-03, University of British Columbia, 2005.
- [96] LANGFORD, J., “Machine Learning Jobs are Growing on Trees.” hunch.net, June 2007.
- [97] LEE, D. and GRAY, A., “Faster Gaussian Summation: Theory and Experiment,” in *Uncertainty in Artificial Intelligence (UAI)*, 2006.
- [98] LEE, D., GRAY, A., and MOORE, A., “Dual-Tree Fast Gauss Transforms,” in *Advances in Neural Information Processing Systems 18* (WEISS, Y., SCHÖLKOPF, B., and PLATT, J., eds.), (Cambridge, MA), pp. 747–754, MIT Press, 2006.
- [99] LIU, H., MOTODA, H., and YU, L., “Feature Selection with Selective Sampling,” in *International Conference on Machine Learning (ICML)*, pp. 395–402, 2002.
- [100] LIU, H., MOTODA, H., and YU, L., “A selective sampling approach to active feature selection,” *Artificial Intelligence*, vol. 159, pp. 49–74, November 2004.
- [101] MACKAY, D. J. C., “Gaussian Processes: A Replacement for Neural Networks?,” NIPS 1997 Tutorial Lecture Notes, 1997.
- [102] MARCH, W. and GRAY, A., “Large-Scale Euclidean MST and Hierarchical Clustering,” in *NIPS Workshop on Efficient Machine Learning*, 2007.
- [103] MCCULLOCH, C. E., “Maximum Likelihood Variance Components Estimation for Binary Data,” *Journal of the American Statistical Association*, vol. 89, no. 425, pp. 330–335, 1994.

- [104] METTU, R. R., *Approximation Algorithms for NP-Hard Clustering Problems*. PhD thesis, The University of Texas at Austin, 2002.
- [105] MOGHADDAM, B., WEISS, Y., and AVIDAN, S., “Spectral Bounds for Sparse PCA: Exact and Greedy Algorithms,” in *Neural Information Processing Systems (NIPS)*, 2005.
- [106] MOORE, A., CONNOLLY, A., GENOVESE, C., GRAY, A., GRONE, L., KANIDORIS II, N., NICHOL, R., SCHNEIDER, J., SZALAY, A., SZAPUDI, I., and WASSERMAN, L., “Fast Algorithms and Efficient Statistics: N-point Correlation Functions,” 2000.
- [107] OSINSKI, S., STEFANOWSKI, J., and WEISS, D., “Lingo: Search results clustering algorithm based on singular value decomposition,” in *Proceedings of the International Conference on Intelligent Information Systems*, 2004.
- [108] PAINTER, J. and SLOAN, K., “Antialiased ray tracing by adaptive progressive refinement,” in *Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)*, 1989.
- [109] PEARL, J., “Evidential Reasoning Using Stochastic Simulation,” *Artificial Intelligence*, vol. 32, pp. 245–257, 1987.
- [110] PECHENIZKIY, M., PUURONEN, S., and TSYMBAL, A., “The Impact of Sample Reduction on PCA-based Feature Extraction for Supervised Learning,” in *ACM Symposium on Applied Computing*, 2006.
- [111] PLATT, J., “Fast Training of Support Vector Machines using Sequential Minimal Optimization,” in *Advances in Kernel Methods: Support Vector Learning* (B. SCHOLKOPF, C. J. C. BURGESS, A. J. S., ed.), ch. 12, MIT Press, 1999.
- [112] PLATT, J., “Using Sparseness and Analytic QP to Speed Training of Support Vector Machines,” in *Advances in Neural Information Processing Systems (NIPS) 11*, 1999.
- [113] PROVOST, F., JENSEN, D., and OATES, T., “Efficient progressive sampling,” in *International Conference on Knowledge Discovery and Data Mining (KDD)*, pp. 23–32, ACM Press, 1999.
- [114] QUINLAN, J. R., “Learning efficient classification procedures and their application to chess end games,” in *Machine Learning. An Artificial Intelligence Approach*, pp. 463–482, Tioga, 1983.
- [115] RAYKAR, V. C. and DURAISWAMI, R., *Large Scale Kernel Machines*, ch. The Improved Fast Gauss Transform with applications to machine learning. MIT Press, 2006.

- [116] RAYKAR, V. C., DURAI SWAMI, R., and KRISHNAPURAM, B., “A fast algorithm for learning large scale preference relations,” in *Conference on Artificial Intelligence and Statistics (AISTATS)*, 2007.
- [117] RAYKAR, V. C. and RAMANI DURAI SWAMI, R., “Fast optimal bandwidth selection for kernel density estimation,” in *SIAM International Conference on Data Mining (SDM)*, 2006.
- [118] RIEGEL, R., GRAY, A., and RICHARDS, G., “Massive-Scale Kernel Discriminant Analysis: Mining for Quasars,” in *SIAM International Conference on Data Mining*, 2008.
- [119] RIGGELSEN, C. and FEELDERS, A., “Learning Bayesian Network Models from Incomplete Data using Importance Sampling,” in *International Conference on AI and Statistics (AISTATS)*, 2005.
- [120] RIOS INSUA, D. and MILLER, P., “Feedforward Neural Networks for Nonparametric Regression,” in *Practical Nonparametric and Semiparametric Bayesian Statistics*, pp. 181–191, Springer-Verlag, 1998.
- [121] ROSENBLATT, M., “Conditional probability density and regression estimators,” in *Multivariate Analysis II* (KRISHNAIAH, P. R., ed.), pp. 25–31, Academic Press, New York, 1969.
- [122] ROWEIS, S. T. and SAUL, L. K., “Nonlinear Dimensionality Reduction by Locally Linear Embedding,” *Science*, vol. 290, pp. 2323–2326, 2000.
- [123] RUBINSTEIN, R. Y., *Simulation and the Monte Carlo Method*. John Wiley & Sons, 1981.
- [124] RUDOY, D. and WOLFE, P. J., “Monte Carlo Methods for Multi-Modal Distributions,” in *Asilomar Conference on Signals, Systems and Computers (ACSSC)*, 2006.
- [125] SARLOS, T., “Improved Approximation Algorithms for Large Matrices via Random Projections,” in *47th IEEE Symposium on Foundations of Computer Science (FOCS)*, pp. 143–152, 2006.
- [126] SCHAPIRE, R. E., STONE, P., MCALLESTER, D., LITTMAN, M. L., and CSIRIK, J. A., “Modeling Auction Price Uncertainty Using Boosting-based Conditional Density Estimation,” in *International Conference on Machine Learning (ICML)*, 2002.
- [127] SCHEFFER, T. and WROBEL, S., “Incremental maximization of non-instance-averaging utility functions with applications to knowledge discovery problems,” in *International Conference on Machine Learning (ICML)*, 2001.

- [128] SCHOLKOPF, B., SMOLA, A. J., and MULLER, K.-R., “Nonlinear Component Analysis as a Kernel Eigenvalue Problem,” *Neural Computation*, vol. 10, no. 5, pp. 1299–1319, 1998.
- [129] SCHRAUDOLPH, N. and GRAEPEL, T., “Towards Stochastic Conjugate Gradient Descent methods,” in *International Conference on Neural Information Processing (ICONIP)*, (Singapore), 2002.
- [130] SCHRAUDOLPH, N., YU, J., and GUENTER, S., “A Stochastic Quasi-Newton Method for Online Convex Optimization,” in *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2007.
- [131] SCHRAUDOLPH, N. N. and GRAEPEL, T., “Combining Conjugate Direction Methods with Stochastic Approximation of Gradients,” in *Workshop on Artificial Intelligence and Statistics (AISTATS)*, 2003.
- [132] SHEN, Y., NG, A. Y., and SEEGER, M., “Fast Gaussian Process Regression using KD-Trees,” in *Neural Information Processing Systems (NIPS)*, 2006.
- [133] SHIGANOV, I. S., “Refinement of the upper bound of the constant in the central limit theorem,” *Journal of Soviet Mathematics*, pp. 2545–2550, 1986.
- [134] SILVERMAN, B. W., *Density Estimation for Statistics and Data Analysis*, vol. 26 of *Monographs on Statistics and Probability*. Chapman & Hall, 1986.
- [135] STENDER, N., *Essays on Marketing Engineering*. PhD thesis, University of Aarhus, Denmark, 2006.
- [136] SUTTON, R. S. and BARTO, A. G., *Reinforcement Learning: An Introduction*. MIT Press, 1998.
- [137] THRUN, S., FOX, D., BURGARD, W., and DELLAERT, F., “Robust Monte Carlo Localization for Mobile Robots,” *Artificial Intelligence*, vol. 128, no. 1-2, pp. 99–141, 2000.
- [138] TOIVONEN, H., “Sampling large databases for association rules,” in *Conference on Very Large Data Bases (VLDB)*, 1996.
- [139] WANG, P., LEE, D., GRAY, A., and REHG, J. M., “Fast Mean Shift with Accurate and Stable Convergence,” in *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2007.
- [140] WASSERMAN, L., *All of Statistics*. Springer, 2005.
- [141] WATKINS, C. I. C. H. and DAYAN, P., “Q-learning,” *Machine Learning*, vol. 8, pp. 279–292, 1992.
- [142] WEI, G. C. G. and TANNER, M. A., “A Monte Carlo Implementation of the EM Algorithm and the Poor Man’s Data Augmentation Algorithms,” *Journal of the American Statistical Association*, vol. 85, no. 411, pp. 699–704, 1990.

- [143] WEINBERGER, K. Q. and TESAURO, G., “Metric Learning for Kernel Regression,” in *Annual Conference on Artificial Intelligence and Statistics (AISTATS)*, 2007.
- [144] WEISS, S. and KULIKOWSKI, C., *Computer Systems that Learn*. Morgan Kaufmann, 1990.
- [145] WILKINSON, D. J. and YEUNG, S. K. H., “Conditional Simulation from Highly Structured Gaussian Systems, with Application to Blocking-MCMC for the Bayesian Analysis of Very Large Linear Models,” *Statistics and Computing*, vol. 12, pp. 287–300, 2002.
- [146] WILSON, D. R. and MARTINEZ, T. R., “The general inefficiency of batch training for gradient descent learning,” *Neural Networks*, vol. 16, pp. 1429–1451, December 2003.