ALGORITHMS AND MECHANISM DESIGN FOR MULTI-AGENT SYSTEMS

A Thesis
Presented to
The Academic Faculty

by

Chinmay Karande

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in Algorithms, Combinatorics and Optimization in the
College of Computing

Georgia Institute of Technology
December 2010

ALGORITHMS AND MECHANISM DESIGN FOR MULTI-AGENT SYSTEMS

Approved by:

Professor Vijay Vazirani, Advisor
College of Computing
Georgia Institute of Technology

Professor Robin Thomas
School of Mathematics
Georgia Institute of Technology

Professor William Cook
School of Industrial and Systems Engineering
Georgia Institute of Technology

Professor Eric Vigoda
College of Computing
Georgia Institute of Technology

Professor Maria-Florina Balcan
College of Computing
Georgia Institute of Technology

Date approved: September 13th 2010

I dedicate this thesis to my paternal Grandfather, Shamrao Karande, for always expecting the best from me, and to my maternal Grandmother, Nalini Rajadhyaksha, for believing that everything I did was the best.

ACKNOWLEDGEMENTS

I would like to thank my advisor Vijay Vazirani, who has been the voice of reason and wisdom during my graduate studies. As a freshman graduate student, I had a variety of interests, but he helped me focus on areas of research where I could succeed. I have always admired the depth of his knowledge and insight, at least some of which, I hope, has rubbed off on me.

The faculty, staff and colleagues at Georgia Tech created a vibrant atmosphere, in which as I look back, the years whizzed by rather quickly. Thanks to Dana Randall and Robin Thomas for helping me join the ACO program, which was the right academic area for me to be in. I will always cherish the time I spent interacting with my collaborators - Nikhil Devanur, Deeparnab Chakrabarty, Gagan Goel, Lei Wang and Pushkar Tripathi. Thanks to encouragement from Vijay, I was able to spend the summers in an industrial research environment at Microsoft and Google. Thanks to my collaborators over there, Amit Aggarwal, Kumar Chellapilla, Reid Andersen, Kamal Jain, Aranyak Mehta and Gagan Aggarwal, for making my time away from school fun and productive.

My parents, Sucheta and Deepak Karande, instilled in me the academic mindset that got me through twenty years of being a student. Thanks to my brother Advait, for being a role model that I look up to. Our education was always the topmost priority for my parents, and I am very grateful for the opportunities they provided us.

Finally, I cannot thank enough, my wife Shefali. She made the decision to throw in her lot with a graduate student, who was then at least four years away from earning a decent wage. She has been both a pillar of support and a fountain of cheer. I hope that with the completion of this thesis, I have repaid her unwavering faith all these years.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

A scenario where multiple entities interact with a common environment to achieve individual and common goals either co-operatively or competitively can be classified as a *Multi-Agent System*. In this thesis, we concentrate on the situations where the agents exhibit selfish, competitive and strategic behaviour, giving rise to interesting game theoretic and optimization problems. From a computational point of view, the presence of multiple agents introduces strategic and temporal issues, apart from enhancing the difficulty of optimization.

We study natural mathematical models of such multi-agent problems faced in practice. We provide approximation algorithms, online algorithms and hardness of approximation results for these problems.

Multi-Agent Submodular Covering Problems. Classical covering problems such as minimum spanning tree, vertex cover and shortest path have been widely used to model a variety of practical situations where the goal is to minimize the cost of a project. However, typically these abstractions do not model two properties commonly observed in the real-world problems: 1) Cost functions observed in practice often exhibit economies of scale and 2) Presence of multiple providers or *agents* each of whom may have different cost function over the same set of objects. The question in general is: How can we incorporate these layers of complexity into combinatorial covering problems?

Submodular functions is a rich and well-studied class of functions used to model economies of scale, or the law of diminishing returns. With this background, we introduce the following class of *combinatorial problems with multi-agent submodular cost functions* - We are given a set of elements X and a collection $C \subseteq 2^X$. There are m agents and every agent i specifies a normalized monotone submodular cost function $f_i : 2^X \to R^+$. The goal is to find a set $S \in C$ and a partition $S_1, ..., S_m$ of S such that $\sum_i f_i(S_i)$ is minimized. The collection C of subsets of X is defined via a combinatorial structure such as a matroid or a graph property

(For example, C can consists of the set of all s-t paths in a graph, yielding a version of the shortest path problem).

We study the following fundamental problems under this multi-agent submodular cost setting: Combinatorial reverse auction, vertex cover, shortest path, minimum spanning tree and minimum perfect matching. We study the approximability of these problems with both algorithmic and hardness results, *i.e.*, upper and lower bounds on the approximation factors.

Combinatorial Auctions with Partially Public Valuations. A central problem in computational mechanism design is that of combinatorial auctions, in which an auctioneer wants to sell a heterogeneous set of items $\mathcal{J}$ to interested agents. Each agent i has a valuation function $f_i(.)$ which describes her valuation $f_i(S)$ for every set $S \subseteq \mathcal{J}$ of items. We consider the case when some inherent property of the items induces a common and publicly known *partial* information about the valuation function of the buyers. In particular, we consider combinatorial auctions where the valuation of an agent i for a set S of items can be expressed as $v_i f(S)$, where v_i is a private single parameter of the agent, and the function f is publicly known. The goal is to design a truthful mechanism which maximizes the social welfare $\sum_i v_i f(S_i)$, where $S_1 \cdots S_n$ is a partition of $\mathcal{J}$.

Our motivation behind studying this problem is two-fold: (a) Such valuation functions arise naturally in the case of ad-slots in broadcast media such as Television and Radio. For an ad shown in a set S of ad-slots, $f(S)$ is, say, the number of *unique* viewers reached by the ad, and v_i is the valuation per-unique-viewer. (b) From a theoretical point of view, this factorization of the valuation function simplifies the bidding language, and renders the combinatorial auction more amenable to better approximation factors.

We present a general technique, based on maximal-in-range mechanisms, that converts any α-approximation non-truthful algorithm ($\alpha \leq 1$) for this problem into $\Omega(\frac{\alpha}{\log n})$ and $\Omega(\alpha)$-approximate truthful mechanisms which run in polynomial time and quasi-polynomial time, respectively.

Online Vertex-weighted Bipartite Matching and Single-Bid Budgeted Allocations. Online bipartite matching is a fundamental problem with numerous applications such as matching candidates to jobs or boys to girls. More recently, this and related problems have received significant attention, because they model the allocation aspect of sponsored search auctions, where multiple agents (advertisers) bid on items (query keywords) which arrive one by one in an online manner. We study the following vertex-weighted online bipartite matching problem: $G(U, V, E)$ is a bipartite graph. The vertices in U have weights and are known ahead of time, while the vertices in V arrive online in an arbitrary order and have to be matched upon arrival. The goal is to maximize the sum of weights of the matched vertices in U. When all the weights are equal, this reduces to the classic *online bipartite matching* problem for which Karp, Vazirani and Vazirani gave an optimal $\left(1 - \frac{1}{e}\right)$-competitive algorithm in their seminal work [38].

Our main result is an optimal $\left(1 - \frac{1}{e}\right)$-competitive randomized algorithm for general vertex weights. Our solution constitutes the first known generalization of the algorithm in [38] in this model and provides new insights into the role of randomization in online allocation problems. It also effectively solves the problem of *online budgeted allocations* [47] in the case when an agent makes the same bid for any desired item, even if the bid is comparable to his budget - complementing the results of [47, 11] which apply when the bids are much smaller than the budgets.

CHAPTER I

INTRODUCTION

A scenario where multiple entities interact with a common environment to achieve individual and common goals either co-operatively or competitively can be classified as a *Multi-Agent System*. In this thesis, we will concentrate on the situations where the agents exhibit selfish, competitive and strategic behaviour, giving rise to interesting game theoretic and optimization problems.

As communication between entities becomes easier, the number of interested parties in any transaction or event is increasing. The rise of the internet, in particular, has catalysed the study of computational problems in multi-agent systems in recent years. We consider natural mathematical models of such multi-agent problems faced in practice. These problems exhibit salient aspects which we outline below, and we study these various aspects in Chapters 2, 3 and 4.

First and foremost, the presence of multiple agents in a system leads to a larger number of possible outcomes or *solutions* to choose from. In a typical optimization scenario, such as minimizing the cost of a project, this makes it possible to construct a cheaper or better solution, but the difficulty of *finding* the best solution obviously increases. We attempt to characterize this effect in the case of some fundamental optimization problems in Chapter 2.

In large multi-agent systems such as the internet, more often than not it is the case that the agents act selfishly, *i.e.* in their own interest. Since the input of the computational problem consists of values reported by various agents, we have to deal with strategic behaviour of the agents, who may report untrue values if they have an incentive to do so in terms of the final outcome. Therefore, even an optimal solution computed using the reported values may be far from optimal on the true values. This interplay between optimization and strategic behaviour is studied in the field of mechanism design. In Chapter 3, we study a mechanism

design problem motivated by advertising on broadcast media such as television and radio.

Finally, all the agents in a system may not be active at the same time. This temporal aspect is studied in the field of online algorithms, wherein only a part of the input is available to the algorithm at any given moment. We study an online allocation problem, motivated by display advertising on the internet, in Chapter 4.

We provide an introduction to the problems studied in this thesis here followed by a more detailed exposition in the respective chapters.

1.1 Multi-Agent Submodular Covering Problems

A multitude of fundamental computational problems with real-world applications can be cast in the following framework: We are given a set X of elements, a collection C of subsets of X (i.e. $C \subseteq 2^X$) and a cost function f over the subsets of X. The collection C is typically specified via a combinatorial structure like a matroid or a graph property (for instance, the set of all spanning trees in a graph). The objective is to select a set $S \in C$ that minimizes $f(S)$.

A major focus in theoretical computer science has been on linear cost functions. However, linear cost functions do not always model the complex dependencies of the costs in the real-world, such as the widely observed *economies of scale*. As a result, even though we might have a good algorithm for solving some linear optimization problem, the output solution can still be suboptimal. Therefore, it is important to extend the study of classical optimization problems to more general cost functions. In this work, we concentrate on submodular cost functions, as they form a rich class and capture the natural properties of economies of scale or the law of diminishing returns.

Another feature that arises in practice is the presence of multiple agents, where each agent has her own cost function. Thus, in the optimal solution, each agent might build only a part of the required combinatorial structure. For example, the Internet is a complex multi-agent system where each service provider owns only a part of the network. For linear cost functions, it is easy to see that having multiple agents does not change the complexity of the original problem. However, this is not the case for more general cost functions and in

particular, for submodular cost functions.

Motivated by these considerations, we define the following class of *combinatorial problems with multi-agent submodular cost functions* (MSCP) - We are given a set of elements X and a collection $C \subseteq 2^X$. There are m agents, and each agent i specifies a normalized monotone submodular cost function $f_i : 2^X \rightarrow R^+$. We assume a *value oracle* model wherein the oracle returns the value $f_i(S)$ of the set S when queried with S and i. The goal is to find a set $S \in C$ and a partition $S_1, ..., S_m$ of S such that $\sum_i f_i(S_i)$ is minimized.

By fixing the collection C to any particular combinatorial structure, one can define a subclass of the problems of interest. In this contribution, we study the following fundamental problems in MSCP :

- **Combinatorial Reverse Auction (CRA)**: We are given a set X of elements and the collection C consists of only the set X i.e. in the required solution all the elements must be covered. This models the situation where a set of jobs needs to be assigned to multiple workers.

- **Submodular Vertex Cover (MS-VC)**: We are given an undirected graph $G(V, E)$. Element set X is the same as the set of vertices V and the collection C consists of all the vertex covers of the graph. Recall that a set $S \subseteq V$ is a vertex cover if every $e \in E$ is incident on a vertex in S.

- **Submodular Shortest Path (MS-SP)**: We are given a connected undirected graph $G(V, E)$, and a pair of vertices $s, t \in V$. Element set X is the same as the set of edges E and the collection C consists of all the paths from s to t.

- **Submodular Minimum Perfect Matchings (MS-MPM)**: We have a undirected graph $G = (V, E)$ with cost functions over E. G contains at least one perfect matching. Element set X is the set of all edges, and the collection C is defined as the set of all perfect matchings of G. Recall that a set $M \subseteq E$ is a perfect matching of G if exactly one edge in M is incident on every vertex.

- **Submodular Minimum Spanning Tree (MS-MST)**: We are given a connected

undirected graph $G = (V, E)$ with cost functions over E. Element set X is the set of all edges, and the collection C is the set of spanning trees of G. Recall that a spanning tree is a minimal connected subgraph of G.

We study the approximability of these problems with both algorithmic and hardness results, *i.e.*, upper and lower bounds on the approximation factors. These results are tabulated in Section 1.4.1.

1.2 Combinatorial Auctions with Partially Public Valuations

A central problem in computational mechanism design is that of combinatorial auctions, in which an auctioneer wants to sell a heterogeneous set of items $\mathcal{J}$ to interested agents. Each agent i has a valuation function $f_i(.)$ which describes her valuation $f_i(S)$ for every set $S \subseteq \mathcal{J}$ of items. In its most general form, the entire valuation function is assumed to be private information which may not be revealed truthfully by the agents. Maximizing the social welfare in a combinatorial auction with an incentive-compatible mechanism is an important open problem. However, recent results [18, 12] have established polynomial lower bounds on the approximation ratio of maximal-in-range mechanisms - which account for a majority of positive results in mechanism design - even when all the valuations are assumed to be submodular. On the other hand, in the non-game-theoretic case, if all the agents' valuations are public knowledge and hence truthfully known, then we can maximize the social welfare to much better factors [19, 20, 59], under varying degree of restrictions on the valuations. In this section, we introduce a model that lies in between these two extremes.

We explore the setting when some inherent property of the items induces a common and publicly known *partial* information about the valuation function of the agents. For instance, in position auctions in sponsored search, the agents' valuation for a position consists of a private value-per-click as well as a public click-through rate, that is known to the auctioneer. Another situation where such private/public factorization of valuations arises is advertisements in broadcast media such as Television and Radio. Suppose we are selling TV ad-slots on a television network. There are m ad-slots and n advertisers interested in them. Let us define a function $f : 2^{[m]} \rightarrow \mathbb{Z}_+$, such that for any set S of ad-slots $f(S)$ is the

number of *unique* viewers who will see the ad if the ad is shown on each slot in S[1]. If an advertiser i is willing to pay v_i dollars per unique viewer reached by her ad, then her total valuation of the set S is $v_i f(S)$.

With this background, we define *single parameter combinatorial auctions with partially public valuations*: We are given a set $\mathcal{J}$ of m items and a global *public* valuation function $f : 2^{\mathcal{J}} \rightarrow \mathbb{R}$. The function f can either be specified explicitly or via an oracle which takes a set S as input and returns $f(S)$. In addition, we have n agents each of whom has a *private* multiplier v_i such that the item set S provides $v_i f(S)$ amount of utility to agent i. The goal is to design a truthful mechanism which maximizes $\sum_i v_i f(S_i)$, where $S_1 \cdots S_n$ is a partition of $\mathcal{J}$.

One can think of this model as combinatorial auctions with *simplified bidding language*. The agents only need to specify one parameter v_i as their bid. Moreover, our problem has deeper theoretical connections to the area of single parameter mechanism design in general. For single parameter domains such as ours, it is known that *monotone* allocation rules characterize the set of all truthful mechanisms. An allocation rule or algorithm is said to be monotone if the allocation parameter of an agent ($f(S_i)$ in our case) is non-decreasing in his reported bid v_i. Unfortunately, often it is the case that good approximation algorithms known for a given class of valuation functions are not monotonic. It is an important and well-known open question in algorithmic mechanism design to resolve whether the design of monotone algorithms is fundamentally harder than the non-monotone ones. In other words, it is not known if, for single parameter problems, we can always convert any α-approximation algorithm into a truthful mechanism with the same factor. We believe that our problem is a suitable candidate to attack this question as it gives a lot of flexibility in defining the complexity of function f. From this discussion, it follows that the only lower bound known for the approximation factor of a truthful mechanism in our setting is the hardness of approximation of the underlying optimization problem.

[1]For a single ad-slot j, the function $f(\{j\})$ is nothing but the television rating for that slot as computed by rating agencies such as Nielsen. In fact, their data collection through set-top boxes results in a TV slot-viewer bipartite graph on the sample population, from which $f(S)$ can be estimated for any set S of ad slots.

We present a general technique, based on maximal-in-range mechanisms, that converts any black-box α-approximation non-truthful algorithm ($\alpha \leq 1$) for this problem into $\Omega(\frac{\alpha}{\log n})$ and $\Omega(\alpha)$-approximate truthful mechanisms which run in polynomial time and quasi-polynomial time, respectively. It is important to note that we do not make any explicit assumptions such as non-negativity or free disposal about the public function f. The black-box algorithm - which is an input - may make some implicit assumptions about f.

1.3 Online Vertex-weighted Bipartite Matching and Single-Bid Budgeted Allocations

Online bipartite matching is a fundamental problem with numerous applications such as matching candidates to jobs or boys to girls. More recently, this and related problems have received significant attention, because they model the allocation aspect of sponsored search auctions, where multiple agents (advertisers) bid on items (query keywords) which arrive one by one in an online manner. A canonical result in online bipartite matching is due to Karp, Vazirani and Vazirani [38], who gave an optimal online algorithm for the unweighted case to maximize the *size* of the matching. In their model, we are given a bipartite graph $G(U, V, E)$. The vertices in U are known ahead of time, while the vertices in V arrive one at a time online in an arbitrary order. When a vertex in V arrives, the edges incident to it are revealed and it can be matched to a neighboring vertex in U that has not already been matched. A match once made cannot be revoked. The goal is to maximize the number of matched vertices.

However, in many real world scenarios, the value received from matching a vertex might be different for different vertices: (1) Advertisers in online display ad-campaigns are willing to pay a fixed amount every time their graphic ad is shown on a website. By specifying their targeting criteria, they can choose the set of websites they are interested in. Each impression of an ad can be thought of as matching the impression to the advertiser, collecting revenue equal to the advertiser's bid. (2) Consider the sale of an inventory of items such as cars. Buyers arrive in an online manner looking to purchase one out of a specified set of items they are interested in. The sale of an item generates revenue equal to the price of the item. The goal in both these cases is to maximize the total revenue.

With this background, we define *Online Vertex-weighted Bipartite Matching*: The input instance is a bipartite graph $G(U, V, E, \{b_u\}_{u \in U})$, with the vertices in U and their weights b_u known ahead of time. Vertices in V arrive one at a time, online, revealing their incident edges. An arriving vertex can be matched to an unmatched neighbor upon arrival. Matches once made cannot be revoked later and a vertex left unmatched upon arrival cannot be matched later. The goal is to maximize the sum of the weights of the matched vertices in U.

Our main result is an optimal $\left(1 - \frac{1}{e}\right)$-competitive randomized algorithm for this problem. Our solution constitutes the first known generalization of the algorithm in [38] in this model and provides new insights into the role of randomization in online allocation problems.

This result also constitutes a step towards the solution of the *online budgeted allocation* problem. This problem was first considered by Mehta *et al* [47] to model the sponsored search auctions: We have n agents and m items. Each agent i specifies a monetary budget B_i and a bid b_{ij} for each item j. Items arrive online, and must be immediately allocated to an agent. If a set S of items is allocated to agent i, then the agent pays the minimum of B_i and $\sum_{j \in S} b_{ij}$. The objective is to maximize the total revenue of the algorithm. An important and unsolved restricted case of this problem is when all the non-zero bids of an agent are equal, *i.e.* $b_{ij} = b_i$ or 0 for all j. Our result effectively solves this case, since it reduces to our vertex-weighted matching problem.

For the general online budgeted allocation problem, no factor better than $\frac{1}{2}$ (achieved by a simple deterministic greedy algorithm [45]) is yet known. The best known lower bound stands at $1 - \frac{1}{e}$ due to the hardness result in [38] for the case when all bids and budgets are equal to 1 - which is equivalent to the unweighted online matching problem. The *small bids* case - where $b_{ij} \ll B_i$ for all i and j - was solved by [47, 11] achieving the optimal $1 - \frac{1}{e}$ deterministic competitive ratio. It was believed that handling *large bids* requires the use of randomization, as in [38], but no generalization of that result was known prior to our work.

Our solution to the vertex-weighted matching problem is a significant step in this direction. Our algorithm generalizes that of [38] and provides new insights into the role of randomization in these solutions. Finally, our algorithm has interesting connections to the

solution of [47] for the *small bids* case - despite the fact that the vertex-weighted matching problem is neither harder nor easier than the *small bids* case. This strongly suggests a possible unified approach to the unrestricted online budgeted allocation problem.

1.4 Contributions, Credits and Organization of the Thesis

1.4.1 Multi-Agent Submodular Covering Problems

In a joint paper with Gagan Goel, Pushkar Tripathi and Lei Wang [25], we gave an approximation algorithm and a matching information theoretic lower bound for each of the subclass of problems[2] that we mentioned in section 1.1. In case of shortest path, minimum spanning tree and minimum perfect matching problems, the bounds established are polynomial and tight upto poly-logarithmic factors. Ignoring these logarithmic factors, we present these results in the table below (Refer to Chapter 2 for proofs). For the reverse auction problem, m is the number of agents and n is the number of items, whereas for all other problems, n is the number of vertices in the instance graph.

	Single-Agent		Multi-Agent	
	Lower bound	Upper bound	Lower bound	Upper bound
Reverse Auction	1	1	$\Omega(\log n)$	$\min(m, \log n)$ [30]
Vertex Cover	$2-\epsilon$	2	$\Omega(\log n)$	$2\log n$
Shortest Path	$\Omega(n^{2/3})$	$O(n^{2/3})$	$\Omega(n^{2/3})$	$O(n)$
Perfect Matching	$\Omega(n)$	n	$\Omega(n)$	n
Spanning Tree	$\Omega(n)$	n	$\Omega(n)$	n

Note that the minimum perfect matching and minimum spanning tree problems, which are polynomial time solvable with linear cost functions, have a large hardness factor with submodular cost functions. We would like to draw attention to our lower bound result for the vertex cover problem in the single agent case. In the classical vertex cover problem, the best known approximation factor is 2, and the best known hardness of approximation is 1.3606 (assuming $P \neq NP$) [16]. Khot et al. [41] showed that achieving a factor of $2-\epsilon$

[2]With the exception of the multi-agent submodular shortest path problem. We comment on this aberration in Section 2.5.2.

'might be' hard by presenting a hardness result based on UGC conjecture [39]. Our results for the single agent submodular vertex cover problem implies that, if the cost function over the set of vertices is submodular, then the optimal approximation factor is indeed 2.

Our hardness results use information theoretic arguments. Our algorithms are based on LP rounding or greedy methods.

We would like to point out that our results for perfect matchings and spanning trees extend to the class of subadditive cost functions, and to related combinatorial structures such as Steiner trees.

Remark: Part of this work also appeared in the PhD thesis of Gagan Goel. Independent of our work, Iwata and Nagano [35] also gave factor 2 approximation algorithm for the single agent submodular vertex cover. They also study submodular cost set cover and submodular edge cover problem.

1.4.2 Combinatorial Auctions with Partially Public Valuations

In a joint paper with Gagan Goel and Lei Wang [26], we presented a general vector fitting technique for designing truthful mechanisms for single parameter combinatorial auctions with partially public valuations. Our main result is a black-box reduction, which accepts any (possibly non-truthful) α-approximation algorithm for our problem as a black-box and uses it to construct a truthful mechanism with an approximation factor of $\Omega\left(\frac{\alpha}{\log n}\right)$. We also give a truthful mechanism with factor $\Omega(\alpha)$ which runs in time $O\left(n^{\log\log n}\right)$. Both these results are corollaries obtained by setting parameters appropriately in Theorem 14 in Chapter 3 to achieve desired trade-off between the approximation factor and the running time. Our results can also be interpreted as converting non-monotone algorithms into monotone ones for the above model.

Our mechanisms are *maximal-in-range*, *i.e.*, they fix a range $\mathcal{R}$ of allocations and compute the allocation $\mathbf{S} \in \mathcal{R}$ that maximizes the social welfare. The technical core of our work lies in careful construction of this range.

While the black-box algorithm may be randomized, our mechanism does not introduce

any further randomization. Depending upon whether the black-box algorithm is deterministic or randomized, our mechanism is deterministically truthful or universally truthful respectively (See Section 3.2 for definitions). The approximation factor of our mechanism is deterministic (or with high probability or in expectation) if the black-box algorithm also provides the approximation guarantees deterministically (or with high probability or in expectation).

Note that we don't need to worry about how the public valuation function f is specified. This is plausible since the function is accessed only from within the black-box algorithm. Hence, our mechanism can be applied to any model of specification - whether it is specified explicitly or through a value or demand oracle - using the corresponding approximation algorithm from that model.

Submodular valuations arise naturally in practice from economies of scale or the law of diminishing returns. Hence, we make a special note of our results when the public valuation is submodular. Using the algorithm of [59] as black-box, our results imply a $\Omega\left(1/\log n\right)$ and $\Omega(1)$ approximation factors in polynomial time and quasi-polynomial time, respectively. In Appendix A.1, we prove with a simple example that the standard greedy algorithm for submodular welfare maximization is not monotone and hence, not truthful. Similarly, the optimal approximation algorithm of [59] is also not known to be non-monotone. For entirely private submodular valuations, the best known truthful mechanism has factor $\Omega(1/\sqrt{m})$ in the *value oracle* model [19] and $\Omega(\log m \log\log m)$ in the *demand oracle* model [17]. Note that the former mechanism is deterministically truthful while the latter is universally truthful.

1.4.3 Online Vertex-weighted Bipartite Matching and Single-Bid Budgeted Allocations

In a joint paper with Gagan Aggarwal, Gagan Goel and Aranyak Mehta [1], we presented an optimal randomized algorithm for the online vertex-weighted bipartite matching problem that is $\left(1-\frac{1}{e}\right)$-competitive in expectation. Our result constitutes the first known generalization of the RANKING algorithm for the unweighted case given by Karp, Vazirani and Vazirani [38] in their seminal work.

Our algorithm, which we call PERTURBED-GREEDY (Refer to Chapter 4), is surprisingly simple to state. The crux of our work lies in a careful counting argument in the probability space, where we bound the weight of bad events (occurrence of a matched vertex) by the weight of the good events (occurrence of an unmatched vertex). PERTURBED-GREEDY and its analysis provides new insights into the role of randomization in online allocation problems, which is important from the point of view of solving more general problems, *viz.* the online budgeted allocation problem.

As we prove in Section 4.5.3, the single bids case of the online budgeted allocation problem reduces to our vertex-weighted matching problem. Therefore our result effectively solves this case, taking a step towards the solution of the general online budgeted allocation problem. Our solution to the single bids case - when all the bids of an agent are equal - has a very interesting 'interface' with the algorithm of Mehta *et al* [47] for the *small bids* case - when all the bids of an agent are very small compared to his budget. This strongly suggests a possible unified approach to the unrestricted online budgeted allocation problem. We elaborate on these implications of our result in Section 4.5.

CHAPTER II

MULTI-AGENT SUBMODULAR COVERING PROBLEMS

In this chapter, we introduce and study combinatorial problems with multi-agent submodular cost functions. We establish upper and lower bound on the approximability of these problems.

Combinatorial problems with multi-agent submodular costs (MSCP): We are given a set of elements X and a collection $C \subseteq 2^X$. There are m agents, and each agent i specifies a normalized monotone submodular cost function $f_i : 2^X \to R^+$. The goal is to find a set $S \in C$ and a partition $S_1, ..., S_m$ of S such that $\sum_i f_i(S_i)$ is minimized.

A function $f : 2^X \to R^+$ is said to be submodular iff for any two sets S and $T \subseteq X$, $f(S) + f(T) \geq f(S \cup T) + f(S \cap T)$. Function f is said to be monotone if $f(S) \leq f(T)$ for any $S \subseteq T$, and normalized if $f(\emptyset) = 0$. Since a submodular function is defined over an exponentially large domain, we will work with the *value oracle* model in which an oracle will return the value of $f(S)$, when queried with the set $S \subseteq X$.

Notice that by fixing the collection C to any particular combinatorial structure, one can define a subclass of the problems of interest. In this contribution, we study the following fundamental problems in MSCP :

- **Combinatorial Reverse Auction (CRA)**: We are given a set X of elements and the collection C consists of only the set X i.e. in the required solution all the elements must be covered. This models the situation where a set of jobs needs to be assigned to multiple workers.
- **Submodular Vertex Cover (MS-VC)**: We are given an undirected graph $G(V, E)$. Element set X is the same as the set of vertices V and the collection C consists of all

the vertex covers of the graph. Recall that a set $S \subseteq V$ is a vertex cover if every $e \in E$ is incident on a vertex in S.

- **Submodular Shortest Path (MS-SP)**: We are given a connected undirected graph $G(V, E)$, and a pair of vertices $s, t \in V$. Element set X is the same as the set of edges E and the collection C consists of all the paths from s to t.

- **Submodular Minimum Perfect Matchings (MS-MPM)**: We have a undirected graph $G = (V, E)$ with cost functions over E. G contains at least one perfect matching. Element set X is the set of all edges, and the collection C is defined as the set of all perfect matchings of G. Recall that a set $M \subseteq E$ is a perfect matching of G if exactly one edge in M is incident on every vertex.

- **Submodular Minimum Spanning Tree (MS-MST)**: We are given a connected undirected graph $G = (V, E)$ with cost functions over E. Element set X is the set of all edges, and the collection C is the set of spanning trees of G. Recall that a spanning tree is a minimal connected subgraph of G.

For each of the above problems, we establish upper and lower bounds on the approximation factor in both the single agent and the multi-agent setting.

2.1 Motivation, Background and Our Results

A multitude of fundamental computational problems with real-world applications can be cast in the following framework: We are given a set X of elements, a collection C of subsets of X (i.e. $C \subseteq 2^X$) and a cost function f over the subsets of X. The collection C is typically specified via a combinatorial structure like a matroid or a graph property (for instance, the set of all spanning trees in a graph). The objective is to select a set $S \in C$ that minimizes $f(S)$.

A major focus in theoretical computer science has been on linear cost functions. The study of combinatorial problems with linear cost functions has led to great developments in the theory of exact and approximation algorithms. However, linear cost functions do not always model the complex dependencies of the costs in a real-world scenario. Submodular

functions form a rich class and capture the natural properties of economies of scale or the law of diminishing returns. As a result, they model cost functions seen in practice more accurately than linear functions.

Another feature that arises in practice is the presence of multiple agents, where each agent has her own cost function. Thus, in the optimal solution, each agent might build only a part of the required combinatorial structure. For example, the Internet is a complex multi-agent system where each service provider owns only a part of the network. For linear cost functions, it is easy to see that having multiple agents doesn't change the complexity of the original problem. However, this is not the case for more general cost functions, such as submodular functions.

In the past, there has been some work along these lines (See [13, 21, 57, 59, 28]), but to the best of our knowledge, none of them has studied multi-agent submodular functions over a truly combinatorial structure. For instance, the work of [13] studies submodular function maximization over matroid constraints in presence of a single agent. The work of [57, 59] considers the multi-agent submodular functions but the combinatorial structure (or the collection C) used in their problem is either the set of all subsets or the whole set itself.

With this background, we propose to extend the algorithmic study of covering problems to the more general model of submodularity and multiple agents. From a practical viewpoint, each of the problem we study is meaningful in its own right. Shortest path and spanning trees are used in network design problems, and it is natural to assume that different agents could have different submodular cost functions depending on the set of edges they can construct cheaply. Similarly, the other problems are used in a variety of situations, especially in relation to algorithmic game theory.

From a theoretical perspective, one would like to extend the tools and techniques developed for combinatorial problems with linear cost functions to as general a setting as possible. Submodular functions are a natural generalization where one would expect to be able to extend the techniques. Despite the significant recent progress on some of the fundamental problems in this area [13, 21, 57, 59, 28], the algorithmic theory for combinatorial problems with submodular cost functions is not substantially developed yet. Many more

basic questions remain to be identified and solved, which could form the basis of tools and techniques for solving more complex problems.

2.1.1 Our Results

We give an approximation algorithm and a matching information theoretic lower bound for each of the subclass of problems that we mentioned earlier[1]. In case of shortest path, minimum spanning tree and minimum perfect matching problems, the bounds established are polynomial and tight upto poly-logarithmic factors. Ignoring these logarithmic factors, we present these results in the table below. For the reverse auction problem, m is the number of agents and n is the number of items, whereas for all other problems, n is the number of vertices in the instance graph.

Table 1: Upper and lower bounds for multi-agent submodular covering problems

	Single-Agent		Multi-Agent	
	Lower bound	Upper bound	Lower bound	Upper bound
Reverse Auction	1	1	$\Omega(\log n)$	$\min(m, \log n)$ [30]
Vertex Cover	$2-\epsilon$	2	$\Omega(\log n)$	$2\log n$
Shortest Path	$\Omega(n^{2/3})$	$O(n^{2/3})$	$\Omega(n^{2/3})$	$O(n)$
Perfect Matching	$\Omega(n)$	n	$\Omega(n)$	n
Spanning Tree	$\Omega(n)$	n	$\Omega(n)$	n

Note that the minimum perfect matching and minimum spanning tree problems, which are polynomial time solvable with linear cost functions, have a large hardness factor with submodular cost functions. We would like to draw attention to our lower bound result for the vertex cover problem in the single agent case. In the classical vertex cover problem, the best known approximation factor is 2, and the best known hardness of approximation is 1.3606 (assuming $P \neq NP$) [16]. Khot *et al* [41] showed that achieving a factor of $2-\epsilon$ 'might be' hard by presenting a hardness result based on UGC conjecture [39]. Our results for the single agent submodular vertex cover problem implies that, if the cost function over the set of vertices is submodular, then the optimal approximation factor is indeed 2.

[1]With the exception of the multi-agent submodular shortest path problem, where a gap remains. We comment on this aberration in Section 2.5.2.

Our hardness results use information theoretic arguments and follow the framework explained in Section 2.2, with some modifications specific to each problem. Our algorithms are based on LP rounding or greedy methods.

We would like to point out that our results for perfect matchings and spanning trees extend to the class of subadditive cost functions, and to related combinatorial structures such as Steiner trees.

Remark: Independent of our work, recently, Iwata and Nagano [35] also gave factor 2 approximation algorithm for the single agent submodular vertex cover. They also study submodular cost set cover and submodular edge cover problem.

2.1.2 Related Work

Submodular functions have been of great interest in optimization in the past. The most fundamental optimization problem concerning submodular functions is, perhaps, the non-monotone submodular function minimization problem. A sequence of papers in this direction [53, 34, 32, 51, 33, 36] has resulted in fast strongly polynomial time combinatorial algorithms. Another related work is that of non-monotone submodular function maximization [21]. Both these algorithms are often used as a subroutine in solving the configuration LPs corresponding to some other submodular combinatorial optimization problem.

Another body of work in optimization over submodular functions deals with welfare maximization [13, 59, 22, 40]. In this context, the reverse auction problem (CRA) that we study, can be thought of as submodular welfare minimization. Calinescu *et al* [13] studied submodular function maximization subject to matroid constraints. They showed that their problem contains as subcases, many other allocation problems, thus giving a unified framework for studying such problems. Matching information theoretic lower bounds were established in [48].

Svitkina and Fleischer [57] studied submodular objective function for problems like sparsest cut, load balancing, and knapsack. They gave $O\left(\sqrt{\frac{n}{\log n}}\right)$ upper and lower bounds for all these problems, showing that all these problems become much harder under submodular costs. For the submodular reverse auctions, where a set of n goods has to be allocated to m

agents (i.e. collection set $C = \{X\}$) with submodular cost functions to minimize the overall cost, a simple greedy algorithm is known to have a factor $\log(n)$ [30]. Goemans *et al* [28] gave an algorithm for constructing explicit approximate submodular functions by querying polynomial number of times to the original submodular function. Some other related work in optimization that uses submodular functions includes [55, 30, 58, 56, 60]. Recently, questions regarding the testability [54] and learnability [6] of submodular functions have been the matter of study.

Recall that the problems we consider in this contribution are very well studied under linear cost functions. Shortest path, perfect matching and spanning tree can be solved exactly in polynomial time. An algorithm for the vertex cover problem with factor 2 for weighted graphs was first given by [7]. The best known hardness of approximation for Vertex Cover is 1.3606 (assuming $P \neq NP$) [16]. Using UGC conjecture [39], Khot and Regev [41] showed that achieving a factor of $2 - \epsilon$ is hard.

2.2 Preliminaries: Information Theoretic Lower Bounds

A problem in our model is said to have information theoretic lower bound of α if any randomized algorithm that approximates the optimum to a factor α with high probability requires super-polynomial number of queries to the value oracle.

By Yao's principle, it suffices to establish the lower bounds for deterministic algorithms acting on an input which is picked randomly from some fixed distribution. To show these approximation gaps, we follow the general framework which was also used in [57, 28, 21]. We will outline this framework in the single agent setting.

The idea is to first choose a problem instance which has a suitably large collection set $C \subseteq 2^X$ of interest. For example, for the minimum spanning tree problem, we choose a graph that has exponentially many spanning trees. Then we design two submodular cost functions f and g. Typically, g is deterministically picked, whereas f is chosen from a distribution. The choice of f and g relies on the following two properties: a) f and g must be 'hard to distinguish' in the sense that they return the same value on almost all queries and b) The optimum values of f and g over C must differ by a large factor. Intuitively this amounts to

'hiding' a particular set $Q \in C$ in f by setting $f(Q)$ to a low value. We employ the following useful construct to achieve this:

Definition 1 (Two-partition function). *A function $f : 2^X \rightarrow \mathbb{R}_+$ is said to be a two-partition function if*

$$f(S) \;=\; |\overline{Q} \cap S| \;+\; \min\{|Q \cap S|,\; r\}$$

where $Q \subseteq X$, $\overline{Q}$ is the complement of Q and r is any constant.

It is easy to verify that a two-partition function is submodular. Such a function *hides* the set Q in the following sense: $f(S)$ differs from $|S|$ only for those sets which have a large enough intersection with Q. On the other hand, by choosing a suitably small value of r, we can ensure large difference between the values of the set Q under f and g. The set Q is chosen from a distribution over C. Since C is designed to be extremely large, this leaves a very small probability of an arbitrary query S made by an algorithm differentiating f from g. By the union bound and a computation path argument [57, 21], an algorithm making polynomially many number of queries cannot distinguish between f and g. Combining this with the gap in the optima of f and g, one proves the lower bound.

We note an important observation from the above discussion, which we will use in our proofs of lower bounds:

Observation 1. *To prove an information theoretic lower bound using two submodular functions f and g with a gap in their optimum values, it suffices to prove that $Pr[\; f(Q) \neq\; g(Q)\;]$ is super-polynomially small over the random choice of $Q \subseteq X$.*

2.3 Combinatorial Reverse Auction

In this problem we are given a set J, of n elements and m agents. For each agent i we have a normalized monotone submodular cost function $f_i : 2^J \rightarrow \mathbb{R}^+$. We wish to partition the elements among the agents to minimize the total cost. We prove a $\Omega(\log n)$ information theoretic hardness result and provide an algorithm that matches this bound. We also prove the same algorithm to be m-approximate. Another $\log n$-approximate algorithm for this problem had previously appeared in [30].

2.3.1 Proof of hardness

As discussed in Section 2.2, the idea is to construct a deterministic instance and a random instance of the CRA so that the optimal solutions of these two instances differ by a factor of $\Omega(\log n)$, and then show that with high probability, a deterministic algorithm which uses only polynomially many value queries can not distinguish between these two instances.

Consider the following deterministic instance of CRA: There are m agents and a set J of $n = m(m+1)^2/4$ elements. The elements are equally partitioned into m blocks $J_1, J_2, ..., J_m$. We will choose m such that $m = 2^d - 1$, for some d. Now each number i between 1 and m can be represented as a vector $\mathbf{a}_i$ in $GF[2]^d$. Let $G_i = \bigcup_{1 \leq k \leq m,\ \mathbf{a}_i \cdot \mathbf{a}_k = 1} J_k$. For each i, $1 \leq i \leq m$, agent i is only interested in elements in G_i. It is easy to see, that G_i consists of those blocks J_k such that a_k differs from a_i in an odd number of bits. Therefore,

$$|G_i| \ = \ \sum_{r \text{ is odd}} {}^dC_r \ = \ \frac{m+1}{2}$$

Thus each agent is interested in only $(m+1)/2$ blocks of elements and for each block there are $(m+1)/2$ agents who are interested in it. Now, we define the cost function $f_i : 2^J \longrightarrow \mathbb{R}^+$ as follows:

$$f_i(S) \ = \ \begin{cases} \min\{|S|, (m+1)^2/4\} & \text{If } S \subseteq G_i \\ \infty & \text{Otherwise} \end{cases}$$

Let us analyze the optimal cost of this instance. We say that an agent is *marked* if the total size of elements assigned to him is at least $(m+1)^2/4$. Among all the optimal solutions, let OPT be the one that maximizes the number of marked agents. We claim that at least d agents are marked in OPT. Suppose not, then without loss of generality, we may assume $M = \{1, 2, ..., t\}$ to be the set of marked agents and $t < d$. The system of linear equations $\mathbf{a}_i \cdot \mathbf{x} = 0, \forall 1 \leq i \leq t$ has at least one solution $\mathbf{x}^* \in GF[2]^d$, since number of equations is less than the number of variables. Let k be the number between 1 and m corresponding to the vector $\mathbf{x}^*$. This implies that no agent in M is interested in block J_k. Let $A_k = \{i_1, i_2, ..., i_w\}$ be the set of agents who are assigned some elements in J_k. Then $A_k \cap M = \emptyset$. Therefore, we can mark one more agent by transferring the elements in J_k from agents $i_2, i_2, ..., i_w$ to

agent i_1 without changing the cost of the new solution. This is a contradiction because of the choice of OPT. Hence, the optimal cost of this instance is at least $(m+1)^2 d/4$.

For the random instance, we have the same sets of agents and elements. Also, each agent is interested in the same set of elements. However, the cost function for each agent is defined by a probability distribution on the set assigned to her. Next we describe our construction of the random cost functions explicitly.

For each element, assign it uniformly at random to one of the agents who is interested in it. Let S_i be the set of elements which agent i gets. Clearly $(S_1, S_2, ..., S_m)$ forms a partition of the element set J. We define the cost function $g_i : 2^J \longrightarrow \mathbb{R}^+$, for agent i as follows:

$$g_i(S) \;=\; \begin{cases} \min\Big(\; |S \cap \overline{S_i}| + \min\Big\{ |S \cap S_i|, (1+\delta)\frac{(m+1)}{2} \; \Big\}, \; \frac{(m+1)^2}{4} \Big) & \text{If } S \subseteq G_i \\ \infty & \text{Otherwise} \end{cases}$$

where $\delta > 0$ is a fixed constant. Notice that we have replaced the $|S|$ from the definition of $f_i(S)$ by a two-partition function (Recall Definition 1).

Now we show that with high probability, a deterministic algorithm using only polynomially many value queries can not distinguish between $\mathbf{f} = (f_1, f_2, ..., f_m)$ and $\mathbf{g} = (g_1, g_2, ..., g_m)$. We prove the following lemma.

Lemma 1. *For any subset S of elements and any i, $1 \leq i \leq m$, $Pr[f_i(S) \neq g_i(S)] = e^{-\Omega(m)}$.*

Proof. Suppose S is a subset of elements and $1 \leq i \leq m$. By our construction, $g_i(S) \leq f_i(S)$. Therefore $Pr[f_i(S) \neq g_i(S)] = Pr[g_i(S) < f_i(S)]$.

First of all, we claim that the above probability is maximized when $S \subseteq G_i$ and $|S| = (m+1)^2/4$. For this, if $S \not\subseteq G_i$, then $f_i(S) = g_i(S) = \infty$ hence $Pr[f_i(S) < g_i(S)] = 0$. Now suppose $S \subseteq G_i$ and $|S| \geq (m+1)^2/4$. Then $f_i(S) = (m+1)^2/4$. Therefore

$$Pr\left[g_i(S) < f_i(S)\right] \;=\; Pr\left[|S \cap \overline{S_i}| + \min\left\{ |S \cap S_i|, (1+\delta)\frac{(m+1)}{2} \right\} \;<\; \frac{(m+1)^2}{4} \right]$$

This probability can only increase when we remove elements from S. For the case when

$|S| \leq (m+1)^2/4$, we get:

$$\begin{aligned} Pr[\ g_i(S) < f_i(S)\ ] &= Pr\left[\ |S \cap \overline{S_i}| + \min\left\{|S \cap S_i|,\ (1+\delta)\frac{(m+1)}{2}\right\} < |S|\ \right] \\ &= Pr\left[\ \min\left\{|S \cap S_i|,\ (1+\delta)\frac{(m+1)}{2}\right\} < |S \cap S_i|\ \right] \\ &= Pr\left[\ |S \cap S_i| > (1+\delta)\frac{(m+1)}{2}\ \right] \end{aligned}$$

Thus, this probability can only increase when more elements are added to S. Hence under the condition $S \subseteq G_i$, $|S| \leq (m+1)^2/4$, the probability is also maximized when $|S| = (m+1)^2/4$.

Now we assume $S \subseteq G_i$ and $|S| = (m+1)^2/4$. In this case, $Pr[g_i(S) < f_i(S)] = Pr[|S \cap S_i| > (1+\delta)(m+1)/2]$, which by a standard Chernoff bound arguments, can be shown to be bounded by $e^{-\Omega(m)}$. □

If we define $\mathbf{f}(S) = (f_1(S), ..., f_m(S))$ and $\mathbf{g}(S) = (g_1(S), ..., g_m(S))$, then by a simple union bound, as a corollary of the lemma, we have $Pr[\mathbf{f}(S) \neq \mathbf{g}(S)] = \text{poly}(m)e^{-\Omega(m)}$. Now suppose $\mathcal{A}$ is a deterministic algorithm which makes polynomially many queries to the value oracle. Then by the union bound, with probability at most $\text{poly}(m) \cdot e^{-\Omega(m)}$, $\mathcal{A}$ can distinguish between $\mathbf{f}$ and $\mathbf{g}$. Notice that for the cost function $\mathbf{g} = (g_1, ..., g_m)$, the optimal solution is at most $(1+\delta)m(m+1)/2$ achieved by assigning S_i to agent i. However, as we showed, the optimal solution for the cost function $\mathbf{f} = (f_1, f_2, ..., f_m)$ has cost at least $d(m+1)^2/4$, thus with high probability, $\mathcal{A}$ can not approximate a CRA instance within factor $\frac{(m+1)^2 d/4}{(1+\delta)m(m+1)/2} \simeq d = c\log n$ for some $c < 1$.

At last, by Yao's principle, we have the following:

Theorem 2. *A randomized approximation algorithm for the **CRA** problem within factor $c\log n$ for some $c < 1$ needs to make exponentially many value queries.*

2.3.2 A $\min(m, \log n)$ approximation algorithm for combinatorial reverse auction

A $\log n$-approximate algorithm for this problem appeared in [30]. In what follows we provide a $\min(m, \log n)$ approximation algorithm. Consider the following LP relaxation (LP1) and its dual (LP2).

$$\min \sum_{S \subseteq V} \sum_{i} x_{i,S} f_i(S) \qquad \text{(LP1)}$$

$$\sum_{S: u \in S} \sum_{i} x_{i,S} \geq 1 \qquad \forall u \in X$$

$$x_{i,S} \geq 0 \qquad \forall S \subseteq V,\ \forall i$$

$$\max \sum_{u \in X} y_u \qquad \text{(LP2)}$$

$$\sum_{u \in S} y_u \leq f_i(S) \qquad \forall S \subseteq V,\ \forall i$$

$$y_u \geq 0 \qquad \forall u \in X$$

In LP1, $x_{i,S}$ is used to represent the fraction of set S that is allocated to agent i. Since $f_i(S) - \sum_{u \in S} y_u$ is a submodular function, we can construct a separation oracle for the dual program using the submodular minimization algorithm as a subroutine. Thus we can solve LP1 and LP2 optimally. The following lemma describes the structure of an optimal solution to LP1.

Lemma 3. *There exists an optimal fractional solution to LP1 such that for every agent i the set $\mathcal{T}_i = \{\ S\ :\ x_{i,S} > 0\ \}$ forms a nested family.*

Proof. Let x be any feasible solution to LP1. If $\mathcal{T}_i$ is not nested, then there exist $A, B \in \mathcal{T}_i$ such that neither A nor B is contained in the other. We may assume $x_{i,A} \geq x_{i,B}$. We will construct another feasible solution x' to LP1 as follows:

- $x'_{i,A \cup B} = x_{i,B}$
- $x'_{i,B} = 0$
- $x'_{i,S} = x_{i,S}$ for all $S \in X$ other than the above.
- $x'_{i,A} = x_{i,A} - x_{i,B}$
- $x'_{i,A \cap B} = x_{i,B}$ if $A \cap B \neq \emptyset$
- $x'_{j,S} = x_{j,S}\ \forall j \neq i$ and $\forall S \in X$

By submodularity, one can verify that the cost of the solution x' is at most the cost of x. If the set $\mathcal{T}'_i$ corresponding to x' is nested, we are done. Otherwise, we repeat the procedure for x'. The termination of the above procedure can be guaranteed by observing that the potential function $\sum_{S \in \mathcal{T}_i} |S|^2$ strictly increases and is polynomially bounded. □

Let $\overline{x}$ be an optimal solution of LP1 with cost W, satisfying the conditions in lemma 3 and $\mathcal{T}_i$ be the corresponding nested families of sets. Let $\mathcal{T} = \bigcup_i \mathcal{T}_i$. Let Y denote the set of uncovered elements in X. In each iteration pick the set $(i, S) \in \mathcal{T}$ minimizing $f_i(S)/|S \cap Y|$. Add S to the cover and assign it to agent i. Remove all the newly covered elements from Y. Repeat until all elements are covered. Since each $\mathcal{T}_i$ is a nested family, an agent can drop

all but the largest set assigned to her. Let (i, S) be the set covering an element u in the integral cover. Then we define $\alpha(u) = f_i(S)/|S \cap Y|$ to be the cost 'borne' by u. Note that $\sum_u \alpha(u)$ is exactly the cost of the integral cover.

Let $u \in X$ be the j'th element to be covered by this algorithm and let (i, S) be the set chosen to cover it. Suppose u was picked during the algorithm. Then since $\overline{x}$ is a fractional cover of Y, $f_i(S)/|S \cap Y| \leq W/|Y|$.

$$\alpha(u) \leq \frac{f_i(S)}{|S \cap Y|} \leq \frac{W}{|Y|} = \frac{W}{(|X| - j + 1)}$$

On the other hand, if u was not picked by the algorithm, then $\alpha(u) \leq W/(|X| - j' + 1) \leq W/(|X| - j + 1)$ for some $j' < j$.

Summing over all u, we conclude that the integral cover has cost at most $W \log n$. To prove that this algorithm is also m-approximate, observe that each set selected has cost at most W. Moreover, each agent is assigned at most one set in the final solution. This proves the claim.

2.4 Vertex Cover

In this section, we consider the submodular vertex cover problem. We first prove an information theoretic lower bound of $2 - \epsilon$ (for any fixed ϵ) for the single agent case and provide an algorithm with approximation ratio of 2. We then present a $2 \log n$ approximation algorithm for the multi-agent case and an information theoretic lower bound of $\Omega(\log n)$.

2.4.1 Single agent case

We are given an undirected graph $G(V, E)$ and a normalized monotone submodular function $f : 2^V \longrightarrow \mathbb{R}$. We wish to find a vertex cover $U \subseteq V$ of graph G such that $f(U)$ is minimized.

Theorem 4. *For every fixed $\epsilon > 0$, any randomized algorithm for the submodular vertex cover problem with an approximation ratio of $2 - \epsilon$ needs exponentially many queries to the value oracle.*

Proof. Consider a bipartite graph $G(A \cup B, E)$ such that $|A| = |B| = n$. The edge set consists of n edges which forms a matching between A and B. Let R be a random minimum

cardinality vertex cover of this graph, which can be picked by choosing one endpoint of every edge uniformly at random.

Define the following two submodular cost functions.

$$\begin{aligned} f_R(S) &= \min\left\{ |S \cap \overline{R}| + \min\left\{ |S \cap R|,\ \frac{(1+\delta)n}{2} \right\},\ n \right\} \\ g(S) &= \min\{ |S|,\ n \} \end{aligned}$$

Here δ is chosen such that $2/(1+\delta) = 2 - \epsilon$. Notice that the optimum value of the vertex cover for the function f_R is $\frac{(1+\delta)n}{2}$, and for g it is n. Thus if we can show that any randomized algorithm, cannot distinguish between f_R and g with high probability, then it will imply an inapproximability ratio of $2/(1+\delta)$ or $2-\epsilon$ for the submodular vertex cover problem.

From Observation 1 it suffices to show that for a deterministic query Q, $Pr[\ f_R(Q) \neq g(Q)\ ]$ is exponentially small, where the probability space is defined over the random choice of set R. Since $f_R(S) \leq g(S)$ for all $S \subseteq V$, $f_R(Q) \neq g(Q)$ implies $f_R(Q) < g(Q)$.

Let Q^* be the optimal query for which $Pr[\ f_R(Q) < g(Q)\ ]$ is maximized. We will show that $|Q^*| = n$. First, suppose that $|Q| \geq n$, then

$$\begin{aligned} Pr[\ f_R(Q) < g(Q)\ ] &= Pr[\ f_R(Q) \ <\ n\ ] \\ &= Pr\left[\ |Q \cap \overline{R}| \ +\ \min\left\{\ |Q \cap R|,\ (1+\delta)\frac{n}{2}\ \right\} \ <\ n\ \right] \end{aligned}$$

which increases as the size of Q is reduced. Thus the size of the optimal query in this case is n.

Now suppose $|Q| \leq n$. In this case,

$$\begin{aligned} Pr[\ f_R(Q) < g(Q)\ ] &= Pr[\ f_R(Q) < |Q|] \\ &= Pr\left[\ |Q \cap \overline{R}| \ +\ \min\left\{\ |Q \cap R|,\ (1+\delta)\frac{n}{2}\ \right\} \ <\ |Q|\ \right] \\ &= Pr\left[\ \min\left\{\ |Q \cap R|,\ (1+\delta)\frac{n}{2}\ \right\} \ <\ |Q \cap R|\ \right] \\ &= Pr[\ |Q \cap R| \ >\ (1+\delta)\frac{n}{2}\ ] \end{aligned}$$

which increases as $|Q|$ is raised. Therefore, the optimal query size in this case is also n.

Hence $|Q^*| = n$. Let k be the number of edges for which both the end points are contained in Q^* and Q_1 be the set of these endpoints ($|Q_1| = 2k$). Let $Q_2 = Q^* - Q_1$. We have:

$$\begin{aligned} Pr[\ f_R(Q^*) < g(Q^*)] &= Pr\left[\ |Q^* \cap R| \ > \ (1+\delta)\frac{n}{2}\ \right] \\ &= Pr\left[\ |Q_2 \cap R| \ > \ (1+\delta)\frac{n}{2} - k\ \right] \\ &= Pr\left[\ |Q_2 \cap R| \ > \ (1+\delta)\frac{|Q_2|}{2} + \delta k\ \right] \qquad (1) \end{aligned}$$

If $\delta k \geq (1-\delta)\frac{|Q_2|}{2}$, then the expression in equation (1) reduces to $Pr[|Q_2 \cap R| > |Q_2|] = 0$. On the other hand if $\delta k < (1-\delta)\frac{|Q_2|}{2}$, then

$$|Q_2| \ = \ n - 2k \ > \ n - \frac{1-\delta}{\delta}|Q_2|$$

which implies $|Q_2| > \delta n$. Every vertex in Q_2 belongs to R with probability $\frac{1}{2}$ with independence, and $E[|Q_2 \cap R|] = |Q_2|/2 = \delta n/2$. Therefore, applying Chernoff bounds:

$$\begin{aligned} Pr[\ f_R(Q^*) < g(Q^*)] &= Pr\left[\ |Q_2 \cap R| \ > \ (1+\delta)\frac{|Q_2|}{2} + \delta k\ \right] \\ &\leq Pr\left[\ |Q_2 \cap R| \ > \ (1+\delta)\frac{|Q_2|}{2}\right] \\ &\leq e^{-\frac{\delta^3 n}{2}} \end{aligned}$$

Hence, the probability that an arbitrary query Q can distinguish between f and g is exponentially small. □

Theorem 5. *There exists an algorithm which finds a 2-approximate solution to the single agent vertex cover problem with submodular costs.*

Proof. We formulate the problem as a configurational LP and round the fractional solution. Let variable x_S be an indicator variable for the set S of vertices being the vertex cover. Then the following LP is a lower bound on the value of the optimal integral solution.

It is not difficult to see that the function $\sum_{v \in S} \sum_{e \in \delta(v)} y_e$ is a modular function. Thus $f(S) - \sum_{v \in S} \sum_{e \in \delta(v)} y_e$ is a submodular function, and we can use the submodular minimization algorithm as a subroutine to construct a separation oracle for the dual. This

$$\begin{aligned} \min \sum_{S\subseteq V} x_S f(S) \qquad & (LP3) \\ \sum_{S:u\in S} x_S + \sum_{S:v\in S} x_S \geq 1 \qquad & \forall (u,v) \in E \\ x_S \geq 0 \qquad & \forall S \subseteq V \end{aligned} \qquad \begin{aligned} \max \sum_{e\in E} y_e \qquad & (LP4) \\ \sum_{v\in S} \sum_{e\in\delta(v)} y_e \leq f(S) \qquad & \forall S \subseteq V \\ y_e \geq 0 \qquad & \forall e \in E \end{aligned}$$

allows us to find an optimal fractional solution to the LP3 with value at most OPT. Let x^* be this solution. Output $Q = \left\{ u \in V \ : \ \sum_{S:u\in S} x^*_S \geq 1/2 \right\}$ as the vertex cover. Clearly, for any $(u,v) \in E$, either $\sum_{S:u\in S} x^*_S \geq 1/2$ or $\sum_{S:v\in S} x^*_S \geq 1/2$ must hold, thus Q is a valid vertex cover of G. Since $2x^*$ is a fractional cover of Q, submodularity implies that $f(Q) \ \leq \ 2\sum_{S\subseteq V} x^*_S f(S) = 2 \cdot \text{OPT}$. □

2.4.2 Multi-Agent Case

We are given an undirected graph $G(V,E)$ and a normalized monotone submodular function $f_i : 2^V \longrightarrow \mathbb{R}$ for each agent i. We wish to find a vertex cover $U \subseteq V$, and a partition $U_1, U_2, \cdots, U_k$ of U such that $\sum_i f(U_i)$ is minimized.

Now we will sketch the proof of the lower bound for the multi-agent case. Consider a suitable instance graph such as the one used in the proof of Theorem 4, and fix a vertex cover Q. For any set S with vertices in $V - Q$, we will set the cost of S very high for every agent. Hence, it will be easy for any algorithm to zero in onto Q as a 'good' vertex cover. However, we can build the same multi-agent cost structure on top of Q, as used the proof of hardness result of reverse auctions (See Section 2.3). Essentially, the problem of finding minimum cost vertex cover then reduces to that of assigning vertices in Q to the various agents so as to minimize the total cost - which is constrained by Theorem 1 to an information theoretic lower bound of $\Omega(\log n)$. Thus we get the following theorem.

Theorem 6. *Any randomized algorithm for the multi-agent submodular vertex cover problem with an approximation ratio $c \log n$ for some constant $c < 1$ needs exponentially many queries to the value oracle.*

$2\log$ n-approximate algorithm: We begin by finding an optimal fraction solution $\overline{x}$ using the LP relaxation LP5, which gives a lower bound on the optimal integral solution.

The given LP can be solved by constructing a separation oracle of the dual program as shown earlier in the single agent case. Consider the set $Q = \left\{ u \in V \ : \ \sum_{S:u\in S} \sum_i \overline{x}_{i,S} \geq 1/2 \right\}$, which forms a valid vertex cover. We will now round $2\overline{x}$ to find an allocation of vertices in Q to the various agents. Let W denote the total cost of the solution $2\overline{x}$.

$$\begin{aligned} \min \sum_{S\subseteq V} \sum_i x_{i,S} f_i(S) \qquad & (LP5) \\ \sum_{S:u\in S} \sum_i x_{i,S} + \sum_{S:v\in S} \sum_i x_{i,S} \geq 1 \qquad & \forall (u,v) \in E \\ x_{i,S} \geq 0 \qquad & \forall S \subseteq V, \ \forall i \end{aligned}$$

At any step of the algorithm let Z contain the uncovered elements in Q. For any fractional cover x of Z, define $\alpha_x(u) = \sum_{S:u\in S} \sum_i \frac{x_{i,S} f_i(S)}{|S \cap Z|}$. Note that $\sum_u \alpha_x(u) = \sum_{i,S} x_{i,s} f_i(S)$. Pick $u \in Z$ that minimizes $\alpha_{2\overline{x}}(u)$. Among the sets containing u, choose a set (i, S) randomly with probability proportional to $x_{i,S}$. Remove all the newly covered elements from Z and iterate until all the elements in Q are covered. Let y denote this integral cover.

Analysis: Let $u_1, u_2, ...$ be the order in which the vertices in Q get covered. We claim that $E[\alpha_y(u_j)] \leq W/(|Q| - j + 1)$. Suppose u_j was picked during the algorithm. Then, $E[\alpha_y(u_j)] \leq \alpha_{2\overline{x}}(u_j)$. Since $2\overline{x}$ covers the remaining $|Q| - j + 1$ elements in Q, $\alpha_{2\overline{x}}(u_j) \leq W/(|Q| - j + 1)$. On the other hand, if u_j was not picked during the algorithm, then

$$E[\alpha_y(u_j)] \ = \ \alpha_{2\overline{x}}(u'_j) \ \leq \ \frac{W}{(|Q| - j' + 1)} \ \leq \ \frac{W}{(|Q| - j + 1)}$$

for some $j' < j$. Summing over j, we have

$$\sum_{i,S} y_{i,S} f_i(S) \ = \ \sum_{u\in Q} \alpha_y(u) \ \leq \ \sum_{u\in Q} \alpha_{2\overline{x}}(u) \ \leq \ W \log n \ \leq \ 2\mathrm{OPT} \cdot \log n$$

This algorithm can be derandomized using standard techniques.

2.5 Shortest Path

In this problem we are given an undirected graph $G = (V, E)$ and a monotone submodular cost function $f_i : 2^E :\longrightarrow \mathbb{R}$ for each agent i. The goal is to find a path P between two given vertices, and partition of P into $P_1, P_2, \cdots, P_k$ such that $\sum_i f_i(P_i)$ is minimized. We first consider the single agent case and provide an information-theoretic lower bound of

$\Omega(n^{2/3})$ for all fixed $\epsilon > 0$, ignoring poly-logarithmic factors. We also present an $O(n^{2/3})$-approximation algorithm for this problem. Lastly, we comment on the gap that exists between the upper and lower bounds for the multi-agent case, in context of our results for the single agent case.

2.5.1 Single agent case

As in previous sections, we proceed by designing two submodular functions that are hard to distinguish in polynomially many queries but have different optimal values. In the general framework outlined in section 2.2, this is accomplished by 'hiding' a random element of lower cost from the target collection C in one of the functions. In this case, C is the set of all $s-t$ paths. However an identical analysis does not work in this case. This is because for a pair of adjacent edges, the events that these edges belong to the random shortest $s-t$ path are are not independent precluding the use of Chernoff bounds which makes the analysis a lot more involved. In this section we use a simple pigeon hole principle argument to solve this problem.

Theorem 7. *Any randomized approximation algorithm for the submodular shortest path problem with factor $O\left(\frac{n^{2/3}}{\log n}\right)$ needs super-polynomially many queries.*

Proof. Consider the graph G which is a level graph having $n^{2/3}+2$ levels of vertices. First level contains only vertex s and the last level contains only t. Each other level has $n^{1/3}$ vertices and there exists a complete bipartite graph between successive levels. Let R be a randomly chosen $s-t$ path of length $n^{2/3}+1$.

Define the following two submodular cost functions $f, g : 2^E \longrightarrow \mathbb{R}^+$:

$$\begin{aligned} f(Q) &= \min\left\{ \ |Q \cap \overline{R}| \ + \ \min\{ \ |Q \cap R|, \ \log n \ \}, \ n^{2/3}+1 \ \right\} \\ g(Q) &= \min\left\{ \ |Q|, \ n^{2/3}+1 \ \right\} \end{aligned}$$

Clearly, the ratio of optima in g and f is $\Omega\left(\frac{n^{2/3}}{\log n}\right)$.

By Observation 1, to prove the lower bound it suffices to prove that $Pr[f(Q) \neq g(Q)]$ is super-polynomially small for an arbitrary query Q. This happens if and only if $f(Q) < g(Q)$.

Making arguments analogous to the proof of theorem 4, $Pr[\ f(Q) < g(Q)\ ]$ is maximized when $|Q| = 1 + n^{2/3}$. Therefore,

$$Pr[\ f(Q) < g(Q)\ ] = Pr[\ |Q \cap R| > \log^2 n\ ]$$

Let E_{even} and E_{odd} be the set of edges which are at distance even and odd respectively from the vertex s. Define $Q_{even} = Q \cap E_{even}$ and $Q_{odd} = Q \cap E_{odd}$. Similarly define R_{even} and R_{odd}. Without loss of generality, let $|Q_{odd}| \geq |Q_{even}|$. Thus,

$$\begin{aligned} Pr[\ |Q \cap R| > \log^2 n\ ] &= Pr[\ |Q_{even} \cap R_{even}| + |Q_{odd} \cap R_{odd}| > \log n\ ] \\ &\leq 2 \cdot Pr[\ |Q_{odd} \cap R_{odd}| > \frac{\log n}{2}\ ] \end{aligned}$$

Note that the edges in R_{odd} were chosen independently at random since R was chosen uniformly at random. Also $E\left[|Q_{odd} \cap R_{odd}|\right] = O(1)$. Thus by Chernoff bounds we conclude that is $Pr\left[\ |Q \cap R| > \log n\ \right] \leq O\left(e^{-\Omega(\log^2 n)}\right)$, which is super-polynomially small. This proves the theorem. □

Theorem 8. *There exists an algorithm which finds an $O(n^{2/3})$ approximate solution to the single agent shortest path problem with submodular costs.*

Proof. We begin with two simple approaches to get an $O(n)$-approximate algorithm. Interestingly, we can combine the two ideas to obtain an $O(n^{2/3})$-approximate algorithm for the problem.

Goemans *et al* [28] address the problem of finding approximate explicit representations for submodular functions. They use an ellipsoidal approximation of the polymatroid of the submodular function $f : 2^X \rightarrow \mathbb{R}_+$ to assign a weight w_e to every element $e \in X$. The approximated cost of a set is then defined as $\hat{f}(S) = \sqrt{\sum_{e \in S} w_e}$. They prove that for all S,

$$\hat{f}(S) \ \leq \ f(S) \ \leq \ \sqrt{|X|}\hat{f}(S)$$

and therefore, $\hat{f}$ can be though of as an explicit approximate representation of f. To solve our submodular shortest path problem, a possible approach would be to find the weights w_e for all the edges of the graph and then find the path P minimizing $\hat{f}(P)$. This can be done in polynomial time, since minimizing $\hat{f}(P)$ is the same as minimizing $\left(\hat{f}(P)\right)^2$, which

just reduces to the shortest path problem with linear costs. This approach yields a $O(\sqrt{E})$ approximation algorithm. For dense graphs this factor can be as bad as $\Omega(n)$. This method can be useful if the given graph has few edges.

Another simple algorithm to get an $O(n)$ approximation for this problem is to 'guess' the cost of the heaviest edge e in the path, use that as a lower bound on OPT. Define *cost* of an edge e as $f(\{e\})$. The algorithm runs in multiple phases. In each phase choose a new edge and drop all edges that weigh more than the given edge and return any $s-t$ path(if it exists) in the pruned graph. We finally select the smallest $s-t$ paths among those returned during the phases. It is easy to see that this is an $O(l)$ approximate algorithm where l is the number of edges in the optimal path. Once again this can be as bad as $O(n)$ for some graphs. This approach can be useful if the given graph is *dense*, since sufficiently dense graphs are known to have small diameter.

The central idea of our algorithm is to decompose the graph in to sparse and dense clusters. Then we use the first approach to account for sparse regions of the graph and deal with the dense regions using ideas from the second approach.

The algorithm runs in multiple phases, where after each phase we output a path. Final solution is the minimum cost path among these paths. Each phase is identified by a unique edge in the edge set, thus there are $|E|$ phases. Following are the steps, in order, which constitutes a single phase.

Pruning Step: Let e be the edge corresponding to the current phase. Delete all edges that weigh more than e. Let G_e denote the pruned graph. The phase terminates prematurely if the s and the t are disconnected in G_e.

Separation Step: In this step we partition the edge set into those that are in dense regions of the graph and those which belong to sparse regions. Successively remove vertices from G_e whose degree in the remaining graph is at most $n^{1/3}$. Also remove the edges incident on these vertices and add them to the set S_e. Continue removing vertices until all the remaining vertices have degree more than $n^{1/3}$. Let R_e be the remaining edges in G_e. Edges in S_e belong to the sparse part of the graph while those in R_e constitute the dense part of the graph.

Search Step: Using the algorithm in [28], we find an explicit representation for the function f restricted to the S_e. Redefine the costs for edges in S_e to be the weights returned by the ellipsoidal approximation subroutine and set the cost of each edge in R_e to be a zero. Treating these edge costs as additive quantities, find the shortest $s-t$ path passing through e. Let P_e be this path.

Compression Step: The path returned by the search step might contain too many edges, which could be bad for the algorithm. In this step, we compress the path P_e by replacing some of its subpaths by smaller paths(in terms of number of edges). For this we analyze the intersection of P_e with every connected component of $G(V, R_e)$. Let H be an arbitrary connected component of $G(V, R_e)$ and let a be the first vertex where P_e enters H and b be the final vertex that it passes through before leaving H for the last time. Replace the subpath of P_e between a and b with the shortest path in H(in terms of the number of edges) connecting them(refer to the figure below). Do this for every connected component of $G(V, R_e)$. Report this modified path as the solution for this phase.

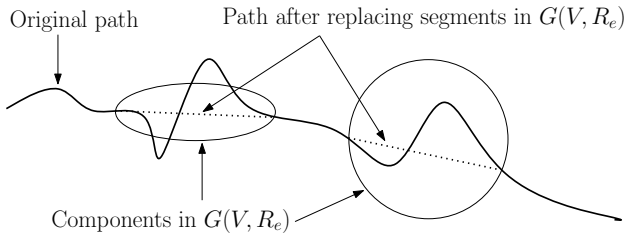


Figure 1: Shortening the path in dense regions

Analysis: To prove that the above algorithm achieves an approximation ratio of $O(n^{2/3})$ we will use the following observation.

Observation 2. *Since all the vertices remaining after the first step have degree greater than $n^{1/3}$, any connected component C of $G(V, R_e)$ has diameter at most $\frac{|V(C)|}{n^{1/3}}$.*

Let P_{OPT} be an optimal path for the problem under the submodular cost function f. Let α be the heaviest edge in this path. Consider the phase corresponding to α. During the separation step of every phase we remove at most $n^{4/3}$ edges since each of the chosen vertices have a degree less than $n^{1/3}$. Thus the subroutine gives an explicit cost function

that is a $O(n^{2/3})$ approximation for all subsets of S_α. Let $\hat{f}(A)$ denote the cost of any $A \subseteq S_\alpha$ returned by the subroutine. Thus for all $A \subseteq S_\alpha$, we have:

$$\hat{f}(A) \leq f(A) \leq n^{2/3}\hat{f}(A) \tag{2}$$

Let P_α be the solution returned by this phase. Define $X_\alpha = P_{OPT} \cap S_\alpha$, $Y_\alpha = P_\alpha \cap S_\alpha$ and $Z_\alpha = P_\alpha \cap R_\alpha$. It follows that,

$$\begin{aligned}
2n^{2/3}f(P_{OPT}) &\geq n^{2/3}f(P_{OPT}) + n^{2/3}f(\{\alpha\}) && (3)\\
&\geq n^{2/3}f(X_\alpha) + n^{2/3}f(\{\alpha\}) && (4)\\
&\geq n^{2/3}\hat{f}(X_\alpha) + n^{2/3}f(\{\alpha\}) && (5)\\
&\geq n^{2/3}\hat{f}(Y_\alpha) + n^{2/3}f(\{\alpha\}) && (6)\\
&\geq n^{2/3}\hat{f}(Y_\alpha) + f(Z_\alpha) && (7)\\
&\geq f(Y_\alpha) + f(Z_\alpha) && (8)\\
&\geq f(Y_\alpha \cup Z_\alpha) && (9)\\
&= f(P_\alpha) && (10)
\end{aligned}$$

Equations (3) and (4) follow from monotonicity. Equation (5) uses equation (2). Equation (6) then follows from equation (5) since P_α is the shortest path under the function $\hat{f}$. Also, using the observation above and summing over all components of $G(V, R_\alpha)$ we conclude that $|Z_\alpha|$ can not be more than $n^{2/3}$. Thus equation (7) follows from equation (6) since each edge in $|Z_\alpha|$ costs at most $f(\{\alpha\})$ and f is submodular. We arrive at equations (8), (9), (10) using equation (2) and the submodularity of function f. □

2.5.2 Multi-Agent case

One glance at Table 1 reveals the fact that the multi-agent submodular shortest path problem is the only problem left 'open' in the sense that we do not have an algorithm that matches the $\Omega(n^{2/3})$ information theoretic lower bound. We will first explain why this gap exists, and then elaborate on some of the interesting issues it emphasizes.

Obviously, the $\Omega(n^{2/3})$ lower bound from Theorem 7 also applies in presence of multiple agents. However, our algorithm from the previous section cannot be ported to the

multi-agent case. Recall the two basic approaches we outlined that yield $O(n)$ approximations to the single agent submodular shortest path problem: 1) Pruning the graph of edges heavier than the heaviest edge in the optimal solution and 2) Ellipsoidal approximation of the submodular function as provided in [28]. While we can still use the former to obtain an $O(n)$-approximation in the multi-agent case, the latter fails due to inherent computational hardness. Finding a path P that minimizes the ellipsoidal approximation function $\hat{f}(P)$ was computationally feasible, because minimizing $\hat{f}(P)$ is equivalent to minimizing $\left(\hat{f}(P)\right)^2$, which reduces to finding the shortest path under linear costs. For this approach to work in the multi-agent case however, we need to find an $s-t$ path P, and partition it into the m agents as $P_1, ..., P_m$ such that $\sum_i \hat{f}_i(P_i)$ is minimized. This function is not linear in terms of the ellipsoidal weights, and in particular is known to be NP-hard to minimize.

The need for a combined computational and communicational lower bound: It is important to note however, that the lower bound established by Theorem 7 is the best possible information theoretic hardness result. Recall that such lower bounds only limit the number of calls made to the value oracle, and no restriction is placed on the computational complexity of the algorithm outside of the oracle calls. Indeed, it is easy to generalize the algorithm in the previous section to the multi-agent case, if we have the computational capacity to minimize the non-linear objective function discussed above. Therefore, there does exist an algorithm that makes polynomially many calls to the value oracle, performs exponentially many other computational operations and guarantees an approximation factor of $O(n^{2/3})$. This indicates that a stronger hardness result needs to combine the computational and information theoretic complexity of the problem into one argument. To our knowledge, such a combined hardness result is not known for any problem.

2.6 Perfect Matching

In this section, we consider the multi-agent submodular minimum perfect matching problem. In this problem we are given a bipartite graph $G(V, E)$ where $|V| = n$, containing at least one perfect matching and a normalized monotone submodular function $f_i : 2^E \longrightarrow R^+$ for each

agent i. We wish to find a perfect matching M, and a partition of M into $M_1, M_2, \cdots, M_m$ such that $\sum_i f_i(M_i)$ is minimized. We first prove an information theoretic lower bound of $\Omega(n)$ on the approximability of the single agent case, which also implies the same bound for the multi-agent case. Then, we give an n-approximate algorithm for the multi-agent case.

As in previous sections, we proceed by designing two submodular functions that are hard to distinguish in polynomially many queries but have widely differing optimal values. In the general framework outlined in section 2.2, this is accomplished by 'hiding' a random element of lower cost from the target collection C in one of the functions. In this case, C is the set of all perfect matchings. Once again choosing a random matching from C however does not serve our purpose because for a fixed pair of edges, the events that these edges belong to the random matching are not independent, thus precluding the use of Chernoff bounds. We circumvent this problem by using the following result from the theory of random graphs [10]:

Lemma 9. *Let $G(n, n, p)$ be a random bipartite graph on $2n$ vertices such that each edge is present independently with probability p. Then*

$$Pr[\ G(n, n, p)\ \textit{contains no perfect matching}\ ] = O(ne^{-np})$$

Now instead of hiding a randomly chosen perfect matching, we hide a collection of randomly and independently chosen edges that contains a perfect matching with high probability. We prove the following theorem.

Theorem 10. *Any randomized approximation algorithm for the submodular minimum cost perfect matching problem with factor $O\left(\frac{n}{\log^2 n}\right)$ needs super-polynomially many queries. There exists an algorithm that approximately finds an n-approximate minimum cost matching in polynomial time.*

Proof. Consider the complete bipartite graph $K_{n,n}$. We choose a random subset R of edges by picking each edge independently with probability $p = \log^2 n/n$. By applying lemma 9, the probability that R does not contain a perfect matching is $O\left(ne^{-\log^2 n}\right)$, which is super-polynomially small.

Define the following two submodular cost functions $f_R, g : 2^E \longrightarrow \mathbb{R}^+$:

$$\begin{aligned} f_R(Q) &= \min\{\ |Q \cap \overline{R}|\ +\ \min\{\ |Q \cap R|,\ (1+\delta)\log^2 n\ \},\ n\ \} \\ g(Q) &= \min\{\ |Q|,\ n\ \} \end{aligned}$$

With probability $1-O(ne^{-\log^2 n})$, R contains a perfect matching and hence the minimum cost of a perfect matching in f is at most $(1+\delta)\log^2 n$. Therefore the ratio of optima in g and f is $\Omega\left(\frac{n}{\log^2 n}\right)$ with high probability.

Now we look at the probability that the algorithm can not distinguish f and g. By Observation 1 it suffices to prove that $Pr[f_R(Q) \neq g(Q)]$ is super-polynomially small for an arbitrary query Q. It's easy to see that $f_R(S) \leq g(S)$, thus these two functions differ on Q if and only if $f_R(Q) < g(Q)$.

Making arguments analogous to the proof of theorem 4, $Pr[\ f_R(Q) < g(Q)\ ]$ is maximized when $|Q| = n$. Therefore,

$$Pr[\ f_R(Q) < g(Q)\ ] = Pr\left[\ |Q \cap R| > (1+\delta)\log^2 n\ \right]$$

Since $E[|Q \cap R|] = \log^2 n$ and edges were picked uniformly at random, we can apply Chernoff bounds to conclude that this probability is $O(e^{-\delta^2 \log^2 n})$. This proves the theorem. □

Factor n approximation algorithm for MS-MPM: We are given a graph $G(V, E)$ and submodular cost functions f_i for each agent. Define a new cost function w over E as $w_e = \min_i f_i(\{e\})$ and define $w(Z) = \sum_{e \in Z} w_e$ for all $Z \subseteq E$. Since w is an additive valuation function we can find a minimum value perfect matching in polynomial time. Let M be such a matching. Assign each edge $e \in M$ to the agent having the minimum valuation for that edge. Let the cost of this solution under the original valuation functions be W.

Analysis: We now prove that this is an n-approximate algorithm. By submodularity we have $W \leq w(M)$. Let M_0 be an optimal solution of MS-MPM having value OPT. Since M is a minimum weight matching under w, $w(M) \leq w(M_0)$.

Let $w_{max} = \max_{e \in M_0}\{f_i(e)\ |\ \text{e assigned to agent } i\ in\ M_0\}$. By submodularity of the cost functions, $w(M_0) \leq n \cdot w_{max}$. By monotonicity we have $w_{max} \leq OPT$. Therefore,

$$W\ \leq\ w(M)\ \leq\ w(M_0)\ \leq\ n \cdot w_{max}\ \leq\ n \cdot OPT$$

This completes the analysis.

2.7 Spanning Tree

In this section, we consider the multi-agent submodular minimum spanning tree problem. We are given a connected graph $G(V, E)$ where $|V| = n$ and a normalized monotone submodular function $f_i : 2^E \longrightarrow R^+$ for each agent i. We want to find a spanning tree T of G, and a partition of T into $T_1, T_2, \cdots, T_m$ such that $\sum_i f_i(T_i)$ is minimized. We first prove an information theoretic lower bound of $\Omega(n)$ on the approximability of the single agent case, which also implies the same bound for the multi-agent case. Then, we give an n-approximate algorithm for the multi-agent case.

To prove the lower bound we will provide two submodular functions that can not be distinguished in polynomially many queries and have widely differing optimal values. As in Section 2.6, we will use the following lemma [10] in the proof.

Lemma 11. *Let $G(n, p)$ be a random graph on n vertices such that each edge is present independently with probability p. Then*

$$Pr[\ G(n,p)\ \textit{is disconnected}\ ] \leq n(1-p)^n.$$

Theorem 12. *Any randomized approximation algorithm for the submodular minimum spanning problem on a with factor $O\left(\frac{n}{\log^2 n}\right)$ needs super-polynomially many queries. There exists an algorithm that approximately finds an n-approximate spanning tree in polynomial time.*

Proof. Consider K_n, the clique graph on n vertices. We choose a random subset of edges R, by picking each edge independently with probability $p = \log^2 n/n$. By applying Lemma 11, the probability that R is not connected is $O\left(ne^{-\log^2 n}\right)$, which is super-polynomially small.

Define the following two submodular cost functions $f_R, g : 2^E \longrightarrow \mathbb{R}^+$:

$$\begin{aligned} f_R(Q) &= \min\left\{\ |Q \cap \overline{R}|\ +\ \min\left\{\ |Q \cap R|,\ (1+\delta)\log^2 n\ \right\},\ n\ \right\} \\ g(Q) &= \min\{\ |Q|,\ n\ \} \end{aligned}$$

With probability $1 - O(ne^{-n^\epsilon})$, R is connected and hence, the cost of the optimal spanning tree in f is at most $(1+\delta)\log^2 n$. Therefore, the ratio of optimal solution values in g and f is $\Omega\left(\frac{n}{\log^2 n}\right)$ with high probability.

Making arguments similar to the proof of Theorem 10, we conclude that $Pr[\ f_R(Q) < g(Q)\ ] = O(ne^{-\delta^2 \log^2 n})$ for any query Q. By observation 1, this suffices to prove the theorem.

Factor n approximation algorithm for MS-MST: We are given a graph $G(V, E)$ and submodular cost functions f_i for each agent. Define a new cost function w over E as $w_e = \min_i f_i(\{e\})$. Run Kruskal's algorithm on G treating w_e as the cost of the edge e to get a minimum spanning tree T. Assign each edge $e \in T$ to the agent i minimizing $f_i(\{e\})$. The proof that this constitutes an n-approximate solution follows similar arguments as the analysis of the n-approximate algorithm for perfect matching. □

2.8 Discussion

The setting that we have considered in this work is quite general, and is a very exciting avenue of research. There are many other interesting problems in this class such as minimum graph cut, edge cover which could be studied in the future work. We have considered the covering problems in this work, one can ask the same questions for packing problems like maximum matching. Extension to multi-agent makes a natural connection to Game Theory. Mechanism design of these combinatorial problems also has interesting applications.

CHAPTER III

COMBINATORIAL AUCTIONS WITH PARTIALLY PUBLIC VALUATIONS

A central problem in computational mechanism design is that of combinatorial auctions, in which an auctioneer wants to sell a heterogeneous set of items $\mathcal{J}$ to interested agents. Each agent i has a valuation function $f_i(.)$ which describes her valuation $f_i(S)$ for every set $S \subseteq \mathcal{J}$ of items. In this chapter, we consider combinatorial auctions where $f_i(.)$ can be expressed as $f_i(S) = v_i f(S)$, where v_i is a private single parameter of the agent, and the function f is publicly known.

Single parameter combinatorial auctions with partially public valuations: We are given a set $\mathcal{J}$ of m items and a global *public* valuation function[1] $f : 2^{\mathcal{J}} \rightarrow \mathbb{R}$. The function f can either be specified explicitly or via an oracle which takes a set S as input and returns $f(S)$. In addition, we have n agents each of whom has a *private* multiplier v_i such that the item set S provides $v_i f(S)$ amount of utility to agent i. The goal is to design a truthful mechanism which maximizes $\sum_i v_i f(S_i)$, where $S_1 \cdots S_n$ is a partition of $\mathcal{J}$.

Our motivation behind studying this problem is two-fold: (a) Such valuation functions arise naturally in the case of ad-slots in broadcast media such as Television and Radio. (b) From a theoretical point of view, this factorization of the valuation function simplifies the bidding language, and renders the combinatorial auction more amenable to better approximation factors.

We present a general technique, based on maximal-in-range mechanisms, that converts

[1]We do not make any explicit assumptions such as non-negativity or free disposal about the function f. We provide a method to convert any non-truthful black-box algorithm into a truthful mechanism. This black-box algorithm may make some implicit assumptions about f.

any α-approximation non-truthful algorithm ($\alpha \leq 1$) for this problem into $\Omega(\frac{\alpha}{\log n})$ and $\Omega(\alpha)$-approximate truthful mechanisms which run in polynomial time and quasi-polynomial time, respectively.

3.1 Motivation, Background and Our Results

Combinatorial auction is a central problem in computational mechanism design an auctioneer wants to sell a set $\mathcal{J}$ of items to interested agents. Each agent i has a valuation function $f_i(.)$ which describes her valuation $f_i(S)$ for every set $S \subseteq \mathcal{J}$ of items. In its most general form, the entire valuation function is assumed to be private information which may not be revealed truthfully by the agents. Maximizing the social welfare in a combinatorial auction with an incentive-compatible mechanism is an important open problem. However, recent results [18, 12] have established polynomial lower bounds on the approximation ratio of maximal-in-range mechanisms - which account for a majority of positive results in mechanism design - even when all the valuations are assumed to be submodular. On the other hand, in the non-game-theoretic case, if all the agents' valuations are public knowledge and hence truthfully known, then we can maximize the social welfare to much better factors [19, 20, 59], under varying degree of restrictions on the valuations. Our model studied in this chapter lies in between these two extremes.

We explore the situation when some inherent property of the items induces a common and publicly known *partial* information about the valuation function of the agents. For instance, in position auctions in sponsored search, the agents' valuation for a position consists of a private value-per-click as well as a public click-through rate, that is known to the auctioneer. Another situation where such private/public factorization of valuations arises is advertisements in broadcast media such as Television and Radio. Suppose we are selling TV ad-slots on a television network. There are m ad-slots and n advertisers interested in them. Let us define a function $f : 2^{[m]} \rightarrow \mathbb{Z}_+$, such that for any set S of ad-slots $f(S)$ is the number of *unique* viewers who will see the ad if the ad is shown on each slot in S[2]. If an

[2]For a single ad-slot j, the function $f(\{j\})$ is nothing but the television rating for that slot as computed by rating agencies such as Nielsen. In fact, their data collection through set-top boxes results in a TV slot-viewer bipartite graph on the sample population, from which $f(S)$ can be estimated for any set S of ad slots.

advertiser i is willing to pay v_i dollars per unique viewer reached by her ad, then her total valuation of the set S is $v_i f(S)$.

One can think of this model as combinatorial auctions with *simplified bidding language*. The agents only need to specify one parameter v_i as their bid. Moreover, our problem has deeper theoretical connections to the area of single parameter mechanism design in general. For single parameter domains such as ours, it is known that *monotone* allocation rules characterize the set of all truthful mechanisms. An allocation rule or algorithm is said to be monotone if the allocation parameter of an agent ($f(S_i)$ in our case) is non-decreasing in his reported bid v_i. Unfortunately, often it is the case that good approximation algorithms known for a given class of valuation functions are not monotonic. It is an important and well-known open question in algorithmic mechanism design to resolve whether the design of monotone algorithms is fundamentally harder than the non-monotone ones. In other words, it is not known if, for single parameter problems, we can always convert any α-approximation algorithm into a truthful mechanism with the same factor. We believe that our problem is a suitable candidate to attack this question as it gives a lot of flexibility in defining the complexity of function f. From this discussion, it follows that the only lower bound known for the approximation factor of a truthful mechanism in our setting is the hardness of approximation of the underlying optimization problem.

3.1.1 Our Results and techniques

We give a general technique which accepts any (possibly non-truthful) α-approximation algorithm for our problem as a black-box and uses it to construct a truthful mechanism with an approximation factor of $\Omega\left(\frac{\alpha}{\log n}\right)$. We also give a truthful mechanism with factor $\Omega(\alpha)$ which runs in time $O\left(n^{\log\log n}\right)$. Both these results are corollaries obtained by setting parameters appropriately in Theorem 14 to achieve desired trade-off between the approximation factor and the running time. Our results can also be interpreted as converting non-monotone algorithms into monotone ones for the above model.

Our mechanisms are *maximal-in-range*, *i.e.*, they fix a range $\mathcal{R}$ of allocations and compute the allocation $\mathbf{S} \in \mathcal{R}$ that maximizes the social welfare. The technical core of our work

lies in careful construction of this range.

While the black-box algorithm may be randomized, our mechanism does not introduce any further randomization. Depending upon whether the black-box algorithm is deterministic or randomized, our mechanism is deterministically truthful or universally truthful respectively (See Section 3.2 for definitions). The approximation factor of our mechanism is deterministic (or with high probability or in expectation) if the black-box algorithm also provides the approximation guarantees deterministically (or with high probability or in expectation).

Note that we don't need to worry about how the public valuation function f is specified. This is plausible since the function is accessed only from within the black-box algorithm. Hence, our mechanism can be applied to any model of specification - whether it is specified explicitly or through a value or demand oracle - using the corresponding approximation algorithm from that model.

Submodular valuations arise naturally in practice from economies of scale or the law of diminishing returns. Hence, we make a special note of our results when the public valuation is submodular. Using the algorithm of [59] as black-box, our results imply a $\Omega\left(1/\log n\right)$ and $\Omega(1)$ approximation factors in polynomial time and quasi-polynomial time, respectively. We would like to note that the standard greedy algorithm for submodular welfare maximization is not monotone (See Appendix A.1 for a simple example) and hence, not truthful. Similarly, the optimal approximation algorithm of [59] is also not known to be non-monotone. For entirely private submodular valuations, the best known truthful mechanism has factor $\Omega(1/\sqrt{m})$ in the *value oracle*[3] model [19] and $\Omega(\log m \log\log m)$ in the *demand oracle*[4] model [17]. Note that the former mechanism is deterministically truthful while the latter is universally truthful.

[3]In the value oracle model, a polytime algorithm must make at most polynomially many queries to the function oracle, which returns $f(S)$ when queried with the set S.

[4]When provided a pricing function $p: [m] \to \mathbb{R}$, the demand oracle returns the set S that maximizes $f(S) - \sum_{j \in S} p(j)$. A polytime algorithm in the demand oracle model is constrained to make at most polynomially many such queries.

3.1.2 Related Work

When agents have a general multi-parameter valuation function, the best known truthful approximation of social welfare in the value oracle model is $\Omega(\sqrt{\log m}/m)$ [31]. Under subadditive valuation functions, [19] gave $\Omega(1/\sqrt{m})$-approximate deterministically truthful mechanism in the value oracle model. It is known that no maximal-in-range mechanism making polynomially many calls to the value oracle can have an approximation factor better than $\Omega(1/m^{1/6})$[18] even for the case of submodular valuation functions. A similar $\Omega(1/\sqrt{m})$ hardness result for maximal-in-range algorithms based on NP $\not\subseteq$ P/poly appears in [12]. See [9] for a comprehensive survey of the results, and [52, 12] for other more recent work. Previous work on the single parameter case of combinatorial auctions have primarily focused on the *single-minded* bidders. In this setting, any bidder i is only interested in single set S_i and has a valuation v_i for it. Lehmann et al. [46] gave a truthful mechanism which achieves an essentially best-possible approximation factor of $\Omega(1/\sqrt{m})$. For other results in single-minded combinatorial auction, see [49, 2]. When the desired set is publicly known and only the valuation is private, [4] gave a general technique which converts any α-approximation algorithm into a truthful mechanism with factor $\alpha/\log(v_{max})$. This result is very much in spirit to our work, however the model and the techniques used are very different. Similarly, [44] present a general framework which uses a gap-verifying linear program as black-box to construct mechanisms that are truthful in expectation. Another example of such a black-box construction is the work of Balcan *et al* [5] who construct universally truthful random sampling auctions when the objective is revenue maximization.

For the non-truthful optimization, we note that our problem is hard up to a constant factor (see [48]) even when all the agents have private value equal to 1 and with common valuation function being submodular. For designing monotone algorithms from non-monotone algorithms in the Bayesian setting, see [29]. We also note that TV ad auctions are in use by Google Inc. (see [50]), although currently they treat the valuations for a set of ad-slots as additive with budget constraints, which yields a multi-parameter auction.

Organization: Section 3.2 provides a brief introduction to mechanism design with a few

concepts relevant to our work. Readers familiar with design of truthful mechanisms can skip to Section 3.3 in which we state some basic properties and assumptions about single parameter combinatorial auctions with partially public valuations. Section 3.4 introduces our vector-fitting technique and in Section 3.4.1, we conduct a warm-up exercise by analyzing a simple mechanism. Section 3.5 presents our main result, a vector-fitting mechanism formalized by Theorem 14.

3.2 Preliminaries

In this section, we will outline the basic concepts in mechanism design relevant to our work.

3.2.1 Truthfulness and Mechanism Design

Mechanism design attempts to address the game-theoretic aspect of optimization problems. Let $\mathcal{A}$ be the set of alternatives, and $u_i(a)$ be the valuation of agent i if alternative $a \in \mathcal{A}$ is picked. In a pure optimization setting, all the functions u_i's are assumed to be known to the auctioneer, and a typical goal is to pick an alternative $a \in \mathcal{A}$ that maximizes $\sum_i u_i(a)$. But from a game-theoretic perspective, the agents may have an incentive to lie about their valuation function u_i, if it leads to a better alternative for them. This kind of strategizing often results in arbitrary behaviour from the agents, leading to a loss in the social welfare. Mechanism design tackles this issue by designing algorithms such that truthfully reporting their true valuation function is the dominant strategy for each agent, *i.e.* given any strategies by all the other agents, reporting one's true function maximizes the utility gained by this agent.

There are three notions of truthfulness that may be applicable:

1. **Deterministic truthfulness**: The mechanism must be deterministic and an agent maximizes her utility by reporting her true valuation, for any valuations of all other agents.

2. **Universal truthfulness**: A universally truthful mechanism is a probability distribution over deterministically truthful mechanisms.

3. **Truthfulness in expectation**: A mechanism is truthful in expectation if an agent maximizes her *expected utility* by being truthful.

Every deterministically truthful mechanism is universally truthful and every universally truthful mechanism is truthful in expectation. Hence, deterministic truthfulness is the strictest notion of truthfulness. As noted earlier, our mechanism may be deterministically or universally truthful depending upon whether the black-box α-approximation algorithm is deterministic or randomized.

3.2.2 Vickrey-Clarke-Grove and Maximal-in-range Mechanisms

The Vickrey-Clarke-Grove (VCG) mechanism is a pivotal result in the field of mechanism design to maximize social welfare. It works as follows: let a^* and a^*_{-i} be the alternatives which maximizes $\sum_j v_j(a)$ and $\sum_{j\neq i} v_j(a)$ respectively. Now define payment p_i of agent i to be $\sum_{j\neq i} v_j(a^*_{-i}) - \sum_{j\neq i} v_j(a^*)$. It is now not difficult to see that with this payment function, it is in best interest of every agent to report their true valuations, irrespective of what others report.

As useful as the VCG mechanism is, it cannot be applied in many scenarios where the underlying problem is hard. Solving the optimization problem approximately doesn't preserve the truthfulness always. To overcome this, maximal-in-range variant of the VCG mechanism is a useful technique which optimizes over a smaller range of allocations. That is, the set of allocations that the mechanism may ever produce - the *range* - is chosen to be a small subset of the space of all allocations. The range is chosen to balance the following trade-off: A larger range can yield better approximation but require greater computational complexity. Note that such a range needs to be defined combinatorially *without* any knowledge of the agents' valuations.

For example, the $\Omega(1/\sqrt{m})$-approximate truthful mechanism from [19] is a maximal-in-range mechanism.

3.3 Notations and Basic Properties

By boldface $\mathbf{v}$, we will denote a vector of private multipliers of the agents, where v_i is the multiplier of agent i. For a constant $\beta \geq 0$, let $\beta\mathbf{v} = (\beta v_1, \beta v_2, ..., \beta v_n)$. By boldface $\mathbf{S}$, we

will denote the vector of allocations, where S_i is the set of items allocated to agent i. We will overload the function symbol f to express the social welfare as: $f(\mathbf{v}, \mathbf{S}) = \sum_i v_i f(S_i)$. An allocation $\mathbf{S}$ is *optimal* for a multiplier vector $\mathbf{v}$ if it maximizes $f(\mathbf{v}, \mathbf{S})$.

We begin by observing two simple properties of our problem and its solutions: *symmetry* and *scale-freeness*. Our problem and its solutions are symmetric, *i.e.*, invariant under relabeling of agents in the following sense: Let $\mathbf{v}$ be any multiplier vector, $\mathbf{S}$ be any allocation and π be any permutation of $[n]$. Let $\mathbf{u}$ and $\mathbf{T}$ be such that $u_i = v_{\pi(i)}$ and $T_i = S_{\pi(i)}$. Then clearly, $f(\mathbf{v}, \mathbf{S}) = f(\mathbf{u}, \mathbf{T})$. The problem and its solutions are also invariant under scaling, since we have $f(\beta\mathbf{v}, \mathbf{S}) = \beta \cdot f(\mathbf{v}, \mathbf{S})$.

The above properties lead us to:

Observation 3. Without loss of generality*, every multiplier vector* $\mathbf{v}$ *has non-increasing entries* $v_1 \geq v_2 \geq ... \geq v_n$ *such that* $\sum_i v_i = 1$.

Given a multiplier vector $\mathbf{v}$, let $\mathrm{A}(\mathbf{v})$ be the optimal allocation for $\mathbf{v}$ and $\mathrm{OPT}(\mathbf{v}) = f(\mathbf{v}, \mathrm{A}(\mathbf{v}))$. Moreover, if $f(\mathbf{v}, \mathbf{S}) \geq \alpha \cdot \mathrm{OPT}(\mathbf{v})$ for some $\alpha \leq 1$ then the allocation $\mathbf{S}$ is said to be α-optimal or α-approximate for $\mathbf{v}$.

We note a simple property of $\mathrm{A}(\mathbf{v})$: Let $\mathbf{v}$ be a multiplier vector with $v_1 \geq v_2 \geq ... \geq v_n$. Let $\mathbf{S}$ be any allocation. If $\mathbf{T}$ is a permutation of $\mathbf{S}$ such that $f(T_1) \geq f(T_2) \geq ... \geq f(T_n)$, then $f(\mathbf{v}, \mathbf{T}) \geq f(\mathbf{v}, \mathbf{S})$. In particular, if $\mathbf{S} = \mathrm{A}(\mathbf{v})$ then $f(S_1) \geq f(S_2) \geq ... \geq f(S_n)$.

Finally, we assume the existence of a poly-time black-box algorithm that computes an α-approximate allocation $\mathrm{B}(\mathbf{v})$ for the multiplier vector $\mathbf{v}$. We express the performance guarantees of our truthful mechanisms in terms of α and other parameters of the problem. Although the output allocation $\mathbf{S}$ of such an algorithm may not obey $f(S_1) \geq f(S_2) \geq ... \geq f(S_n)$, it is easy to construct a non-decreasing permutation of $\mathbf{S}$ which only improves the objective function value, as discussed above.

Observation 4. Without loss of generality*, any allocation* $\mathbf{S}$ *output by the black-box algorithm obeys* $f(S_1) \geq f(S_2) \geq ... \geq f(S_n)$.

Henceforth, we enforce assumptions from Observation 3 and 4.

Definition 2 (u dominates w). *We say that a multiplier vector* **u** *dominates* **w** *if there exists an index i such that for $k < i$, $u_k \geq w_k$ and for $k \geq i$, $u_k \leq w_k$.*

Lemma 13. *If* **u** *dominates* **w**, *then* $f(\mathbf{u}, \mathbf{S}) \geq f(\mathbf{w}, \mathbf{S})$ *for any allocation* **S** *satisfying* $f(S_1) \geq f(S_2) \geq ... \geq f(S_n)$.

Proof. For $k < i$, let $x_k = u_k - w_k$. Similarly for $k \geq i$, $y_k = w_k - u_k$. Then

$$\sum_{k=1}^{i-1} x_k - \sum_{k=i}^{n} y_k \;=\; \sum_{k=1}^{n} u_k - \sum_{k=1}^{n} w_k \;=\; 0$$

which means $\sum_{k=1}^{i-1} x_k = \sum_{k=i}^{n} y_k$. Since $f(S_{k_1}) \geq f(S_{k_2})$ whenever $k_1 < i \leq k_2$,

$$f(\mathbf{u}, \mathbf{S}) - f(\mathbf{w}, \mathbf{S}) \;=\; \sum_{k=1}^{i-1} x_k f(S_k) - \sum_{k=i}^{n} y_k f(S_k) \;\geq\; 0$$

□

Staircase Representation: Suppose we represent a multiplier vector **v** as a histogram, which consists of n vertical bars corresponding to $v_1, ..., v_n$, in that order from left to right. Since multiplier vectors have non-increasing components, such a histogram looks like a staircase descending from left to right (Refer to Figure 2 for an example). We will refer to it as the *staircase representation* of **v** and use it mainly as a visual tool.

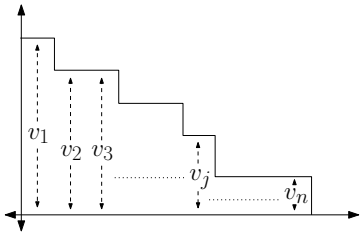


Figure 2: The staircase representation of $\mathbf{v} = (v_1, ..., v_n)$.

3.4 Vector-Fitting Mechanisms

Consider the following candidate approach to single parameter combinatorial auctions with partially public valuations: Fix a set $\mathcal{U}$ of some multiplier vectors. Using the black-box algorithm, compute an α-approximate allocation $\mathrm{B}(\mathbf{v})$ for each vector $\mathbf{v} \in \mathcal{U}$ and populate

the range $\mathcal{R} = \{\ B(\mathbf{v})\ :\ \mathbf{v} \in \mathcal{U}\ \}$. Run the maximal-in-range mechanism which given a multiplier vector $\mathbf{v}$, chooses the allocation $\mathbf{S} \in \mathcal{R}$ that maximizes $f(\mathbf{v}, \mathbf{S})$.

Let's consider the merits and demerits of this mechanism. If the input multiplier vector happens to be in $\mathcal{U}$, then the mechanism will indeed return an output allocation that is at least α-approximate. But we have no guarantees otherwise. If $\mathcal{U}$ consisted of all possible vectors, we would have an α-approximate truthful mechanism that could be computationally infeasible due to the size of $\mathcal{U}$. We handle this trade-off with *vector-fitting*. The intuition behind vector-fitting is as follows: If two multiplier vectors $\mathbf{u}$ and $\mathbf{v}$ are 'very similar' to each other, then B($\mathbf{u}$) and B($\mathbf{v}$) should be 'similar' as well. In particular, B($\mathbf{u}$) should be a reasonably good allocation for $\mathbf{v}$ and vice versa.

Our mechanism will be the same as the candidate mechanism outlined above, except that we will construct the set of vectors $\mathcal{U}$ very carefully. For any input vector of multipliers $\mathbf{v}$, we will guarantee that a reasonably similar vector $\mathbf{v}'$ can be found in $\mathcal{U}$, and hence and allocation $\mathbf{S}'$ is in the range $\mathcal{R}$ with provably large objective value $f(\mathbf{v}, \mathbf{S}')$.

3.4.1 A Simple $\frac{\alpha}{\ln n}$-factor Mechanism

In this section, we will conduct a warm-up exercise by applying the vector-fitting method to construct a simple $\frac{\alpha}{\ln n}$-factor truthful mechanism. Recall that the vector-fitting method as outlined in Section 3.4 starts with a set $\mathcal{U}$ of multiplier vectors. Our set $\mathcal{U}$ is defined as $\mathcal{U} = \{\ \mathbf{u}^j\ :\ 1 \leq j \leq n\ \}$ where $\mathbf{u}^j$ is defined as follows:

$$u_i^j = \frac{1}{j} \text{ for } 1 \leq i \leq j, \text{ zero elsewhere}$$

As before, for each $\mathbf{v} \in \mathcal{U}$, we compute an α-approximate allocation B($\mathbf{v}$) and populate the range $\mathcal{R}$ with it.

Let $\mathbf{v}$ be the input multiplier vector. Let $r_j = \sum_{k=1}^{j} v_j$ for $1 \leq j \leq n$ be the prefix sums of $\mathbf{v}$. We define prefix vectors $\mathbf{d}^j$ of $\mathbf{v}$ as:

$$d_i^j = \frac{v_i}{r_j} \text{ for } 1 \leq i \leq j, \text{ zero elsewhere}$$

It is easy to verify that $\mathbf{d}^j$ is a valid multiplier vector *i.e.*, it has non-increasing components and unit l_1 norm.

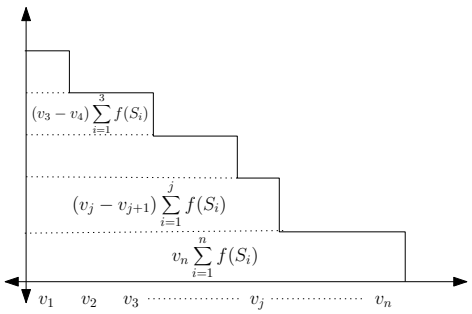


Figure 3: Expressing $f(\mathbf{v}, \mathbf{S})$ as horizontal cuts of the staircase.

Let $\mathbf{S} = \mathrm{A}(\mathbf{v})$ be the optimal allocation for $\mathbf{v}$ and $\mathbf{T}$ be the allocation returned by our mechanism. For notational convenience, define $v_{n+1} = 0$. We start with $\mathrm{OPT}(\mathbf{v}) = f(\mathbf{v}, \mathbf{S})$ and look at how horizontal sections under the staircase of $\mathbf{v}$ contribute to it. See figure 3.

$$
\begin{aligned}
\mathrm{OPT}(\mathbf{v}) &= \sum_{i}^{n} v_i f(S_i) \\
&= v_n \sum_{i=1}^{n} f(S_i) \;+\; (v_{n-1} - v_n) \sum_{i=1}^{n-1} f(S_i) \;+\; \ldots \;+\; (v_1 - v_2) f(S_1) && (11) \\
&= \sum_{j=1}^{n} \left[j(v_j - v_{j+1}) \sum_{i=1}^{j} \frac{f(S_i)}{j} \right] \\
&= \sum_{j=1}^{n} \left[j(v_j - v_{j+1}) \cdot f(\mathbf{u}^j, \mathbf{S}) \right] \\
&\leq \sum_{j=1}^{n} \left[j(v_j - v_{j+1}) \cdot \mathrm{OPT}(\mathbf{u}^j) \right] \\
&\leq \sum_{j=1}^{n} \left[\alpha^{-1} j(v_j - v_{j+1}) \cdot f(\mathbf{u}^j, \mathrm{B}(\mathbf{u}^j)) \right] \\
&\leq \sum_{j=1}^{n} \left[\alpha^{-1} j(v_j - v_{j+1}) \cdot f(\mathbf{d}^j, \mathrm{B}(\mathbf{u}^j)) \right] && (12) \\
&\leq \sum_{j=1}^{n} \left[\frac{j(v_j - v_{j+1})}{\alpha r_j} \cdot f(\mathbf{v}, \mathrm{B}(\mathbf{u}^j)) \right] && (13) \\
&\leq \sum_{j=1}^{n} \left[\frac{j(v_j - v_{j+1})}{\alpha r_j} \cdot f(\mathbf{v}, \mathbf{T}) \right] && (14) \\
&= \frac{f(\mathbf{v}, \mathbf{T})}{\alpha} \left[1 + \sum_{j=2}^{n} \frac{v_j (r_j - j v_j)}{r_j r_{j-1}} \right] && (15)
\end{aligned}
$$

Equation (11) decomposes $f(\mathbf{v}, \mathbf{S})$ as the horizontal cuts of the staircase of $\mathbf{v}$ (See Figure 3). (12) follows from the previous step by applying Lemma 13 to $\mathbf{d}^j$ and $\mathbf{u}^j$. Equation (13)

follows from (16) below, which is a simple restatement.

$$\sum_{i=1}^{j} \frac{v_i}{r_j} \cdot f(S_i) \;\leq\; \frac{1}{r_j} \sum_{i=1}^{n} v_i f(S_i) \;=\; \frac{f(\mathbf{v}, \mathbf{S})}{r_j} \tag{16}$$

Equation (15) is derived from the previous expression by simply rearranging the terms. Since $v_j \leq r_j/j$ and $r_j - jv_j \leq r_{j-1}$, we conclude that

$$1 + \sum_{j=2}^{n} \frac{v_j(r_j - jv_j)}{r_j r_{j-1}} \;\leq\; \sum_{j=1}^{n} \frac{1}{j} \;\leq\; \ln n$$

and

$$f(\mathbf{v}, \mathbf{T}) \geq \alpha \cdot \mathrm{OPT}(\mathbf{v}) / \ln n$$

An example that achieves the bound: We can differentiate each term of the summation in equation (15) to compute the values of v_j for which the term is maximized, so as to make the bound as loose as possible. Surprisingly, a single multiplier vector maximizes all the terms simultaneously! This vector is defined as $v_j = (\sqrt{j} - \sqrt{j-1})/\sqrt{n}$. Some calculations prove that for this multiplier vector, the summation is indeed $\Omega(\ln n)$.

3.5 The Main Result

In this section, we will use vector-fitting to obtain a general technique to convert a non-truthful approximation algorithm for single parameter combinatorial auctions into a truthful mechanism. This technique yields a range of trade-offs between the approximation factor and the running time of the algorithm. We will prove the following theorem:

Theorem 14. *There exists a truthful mechanism for maximizing welfare in a single parameter combinatorial auction with partially public valuations that runs in time* $O((\log_a n)^{\log_b n} \cdot \mathrm{poly}(m, n))$ *and produces an allocation with total welfare at least* $\frac{3\alpha}{4ab} \cdot \mathrm{OPT}(\mathbf{v})$ *- where* α *is the approximation factor of the black-box optimization algorithm and* $a, b > 1$ *are parameters of the mechanism.*

Setting $a = b = 2$ we get: (Henceforth, all logarithms are to base 2)

Corollary 15. *There exists a* $\frac{3\alpha}{16}$*-factor truthful mechanism running in time* $O(n^{\log \log n} \cdot \mathrm{poly}(m, n))$*, i.e. quasi-polynomial time.*

Similarly, setting $a = 2$ and $b = \log n$ we get:

Corollary 16. *There exists a truthful mechanism with factor* $\Omega\left(\frac{\alpha}{\log n}\right)$ *and polynomial running time.*

When the public valuation f is submodular, we have $\alpha = \left(1 - \frac{1}{e}\right)$ and the above corollaries yield factors $\Omega(1)$ and $\Omega\left(\frac{1}{\log n}\right)$ respectively.

3.5.1 Constructing the Range $\mathcal{R}$

Overview: Recall the staircase representation of a multiplier vector $\mathbf{v}$, such as in Figure 2. Depending upon the entries of $\mathbf{v}$, the steps of the staircase may have varying heights. We can construct a discretization of the space of all multiplier vectors by restricting the values the height of any step can take. That is, we populate the initial set $\mathcal{U}$ with all vectors whose components take values of the form b^{-k} for some constant $b > 1$ and for all $k \geq 0$. Now given any input vector $\mathbf{v}$, we can find a vector $\mathbf{u} \in \mathcal{U}$ such that u_i is at most a multiplicative factor b away from v_i. Thus, $\mathbf{u}$ can serve as a vector 'similar' to $\mathbf{v}$. We need more complex machinery to ensure that the size of $\mathcal{U}$ does not blow up, and that the vectors in $\mathcal{U}$ still have unit norm.

Let $a, b > 1$ be suitably chosen parameters of the mechanism. Let $Q = \{\ b^{-k}\ :\ 0 \leq k < \log_b n\ \}$ be a set of values discretizing the interval $(\frac{1}{n}, 1]$ and q be the minimum element of Q. For a multiplier $v_i \geq q$, we define $\lfloor v_i \rfloor$ to be the largest element of Q that is no greater than v_i. For a multiplier vector $\mathbf{v}$ we define the floor of $\mathbf{v}$, $\lfloor \mathbf{v} \rfloor$ as follows:

Definition 3 (Floor $\lfloor \mathbf{v} \rfloor$). *The floor* $\lfloor \mathbf{v} \rfloor$ *of a multiplier vector* $\mathbf{v}$ *is the vector* $\mathbf{u}$ *constructed by Algorithm 1.*

In short, to find the 'floor' of a multiplier vector, we successively round down the 'large' components into elements of Q, until we need to set all the remaining components equal due the monotonicity and unit norm requirement or only 'small' components are remaining. When represented as a staircase (Refer Figure 4), all the steps of $\lfloor \mathbf{v} \rfloor$ except the last one must have height that belongs to Q.

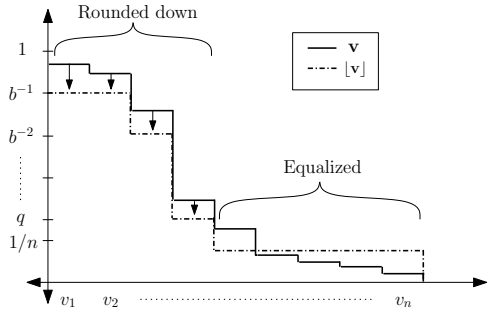


Figure 4: Vertical fitting of $\mathbf{v}$.

Algorithm 1: ConstructFloor

for $i \;=\; 1$ *to* n **do**

$$r \;\leftarrow\; \frac{\left(1 - \sum_{k=1}^{i-1} u_k\right)}{(n-i+1)};$$

`/* r is the minimum permissible value of` u_i `due to monotonicity. */`

if $v_i \;\geq\; q$ *and* $\lfloor v_i \rfloor \;>\; r$ **then**

$u_i \;\leftarrow\; \lfloor v_i \rfloor$;

else

for $j \;=\; i$ *to* n **do**

$u_j \;\leftarrow\; r$;

break

Observation 5. *The floor of a vector* $\mathbf{v}$ *is a valid multiplier vector itself, i.e. it has non-increasing components and unit* l_1 *norm. Moreover,* $\mathbf{v}$ *dominates* $\lfloor \mathbf{v} \rfloor$.

Proof. The procedure to compute $\mathbf{u} = \lfloor \mathbf{v} \rfloor$ easily ensures the unit l_1 norm. Now to prove monotonicity by contradiction, assume that there exists i such that $u_i < u_{i+1}$. Since $\mathbf{v}$ satisfies monotonicity, this can only happen if v_i was strictly rounded down to u_i and v_{i+1} was not. Therefore

$$1 - \sum_{k=1}^{i-1} u_k \;=\; \sum_{k=i}^{n} u_k \;=\; u_i + \sum_{k=i}^{n} u_k \;=\; u_i + (n-i)u_{i+1} > \;(n-i+1) \cdot u_i$$

This implies

$$\lfloor v_i \rfloor \;=\; u_i \;<\; \frac{1 - \sum_{k=1}^{i-1} u_k}{n-i+1}$$

which is impossible since the right-hand side of the above inequality is the minimum value u_i could have been assigned.

To see that $\mathbf{v}$ dominates $\mathbf{u} = \lfloor \mathbf{v} \rfloor$, observe that if the index i does not belong to the last step of $\mathbf{u}$, then v_i must have been rounded down to u_i, and therefore, $u_i \leq v_i$. Now consider the smallest i such that $u_i > v_i$. Then i must belong to the last step of $\mathbf{u}$, and hence $u_j = u_i > v_i \geq v_j$ for any $j \geq i$. □

Intuitively, the floor of a vector is (in a sense formalized by Lemma 17) 'similar' to the vector, and the similarity is parametrized by b.

Lemma 17. *For any multiplier vector* $\mathbf{v}$ *and allocation* $\mathbf{S}$, $f(\lfloor \mathbf{v} \rfloor, \mathbf{S}) \geq \frac{3}{4b} \cdot f(\mathbf{v}, \mathbf{S})$.

Proof. Define $\mathbf{u} = \lfloor \mathbf{v} \rfloor$. Let p be the highest index such that v_p is rounded down by the procedure that constructs $\mathbf{u}$, *i.e.* $u_p = \lfloor v_p \rfloor$ and $u_p > r = u_{p+1}$. Since, $\sum_{i=1}^{p} u_i \leq \sum_{i=1}^{p} v_i$, it is clear that $p < n$. Now for $i \leq p$, we have $u_i = \lfloor v_i \rfloor \geq v_i/b$. Consider two cases about v_{p+1}:

Case 1 - $v_{p+1} \geq q$: In this case, $u_{p+1} = r \geq \lfloor v_{p+1} \rfloor \geq v_{p+1}/b$. For $i \geq p+1$, we have $v_i \leq v_{p+1}$ and $u_i = u_{p+1}$ implying $u_i \geq v_i/b$. Therefore,

$$f(\mathbf{u}, \mathbf{S}) = \sum_{i=1}^{n} u_i f(S_i) \geq \frac{1}{b} \sum_{i=1}^{n} v_i f(S_i) = \frac{1}{b} \cdot f(\mathbf{v}, \mathbf{S})$$

Case 2 - $v_{p+1} < q$: Let $h = \sum_{i=1}^{p} v_i$ and $H = \left(\sum_{i=1}^{p} v_i f(S_i) \right) / f(\mathbf{v}, \mathbf{S})$. From the monotonicity of $\mathbf{S}$, we conclude that

$$H \cdot f(\mathbf{v}, \mathbf{S}) = \sum_{i=1}^{p} v_i f(S_i) \geq h \cdot f(\mathbf{v}, \mathbf{S})$$

and hence $H \geq h$.

Since $u_i \leq v_i$ for all $i \leq p$, and both $\mathbf{u}$ and $\mathbf{v}$ must have unit l_1 norm, we have $\sum_{i>p} u_i \geq \sum_{i>p} v_i = (1-h)$. Hence, $u_i \geq \frac{1-h}{n}$ for $i > p$. By definition, $v_i < q \leq \frac{b}{n}$ for $i > p$. Together, these imply $u_i \geq (1-h)v_i/b$. Finally, using $H \geq h$, we conclude

$$\sum_{i>p} u_i f(S_i) \geq \frac{1-h}{b} \left(\sum_{i>p} v_i f(S_i) \right) \geq \frac{1-H}{b} \left[(1-H) f(\mathbf{v}, \mathbf{S}) \right]$$

Combining these pieces together, we get:

$$\begin{aligned} f(\lfloor \mathbf{v} \rfloor,\ \mathbf{S}) &= \sum_{i=1}^{p} u_i f(S_i) \ + \ \sum_{i>p} u_i f(S_i) \\ &\geq \frac{1}{b} \sum_{i=1}^{p} v_i f(S_i) \ + \ \frac{(1-H)^2}{b} \cdot f(\mathbf{v}, \mathbf{S}) \\ &= \frac{H + (1-H)^2}{b} \cdot f(\mathbf{v}, \mathbf{S}) \ \geq \ \frac{3}{4b} \cdot f(\mathbf{v}, \mathbf{S}) \end{aligned}$$

□

We will construct our preliminary set of vectors $\mathcal{U}'$ as

$$\mathcal{U}' \ = \ \{ \ \mathbf{u} \ : \ \mathbf{u} = \lfloor \mathbf{v} \rfloor \text{ for some multiplier vector } \mathbf{v} \ \}$$

It turns out that $\mathcal{U}'$ is too large for our purposes. Hence we construct a subset $\mathcal{U} \subseteq \mathcal{U}'$, which is small enough. Referring back to the staircase representation of a multiplier vector (Figure 4), we constructed $\mathcal{U}'$ by discretizing the 'height' of each step - by fitting the vectors vertically. Since rounding down the components of $\mathbf{v}$ might lead to many components of $\mathbf{u} = \lfloor \mathbf{v} \rfloor$ having the same value, $\mathbf{u}$ also looks like a staircase, perhaps with 'wider' steps. Each step of $\mathbf{u}$ may have any integral width - at most n.

We construct $\mathcal{U}$ from $\mathcal{U}'$ by further restricting how wide a step can be - by horizontal fitting (See Figure 5). We allow each step (except the last) to be of width $\lceil a^k \rceil$ for some integer $k \geq 0$ - where $a > 1$ is a suitably chosen parameter of the mechanism. To this end, we need to slightly formalize the staircase representation of a multiplier vector, which till now we only used as a visual aid. By a *step* of the staircase of $\mathbf{v}$, we will mean a maximal interval $[i_1, ..., i_2] \subseteq [1, ..., n]$ such that $v_{i_1} = v_{i_2}$. All the indices $i_1 \leq i \leq i_2$ will be said to belong to the step, whereas i_1 and i_2 and the first and last indices of the step. The *height* of the step is given by v_{i_1} and the *width* by $i_2 - i_1 + 1$.

Remark: Notice that just as a multiplier vector can be specified by the n-tuple $(v_1, ..., v_n)$, it can also be identified by specifying the height and width of each step of its staircase representation. In fact, specifying all but the last step of a staircase fixes the last step due to the unit norm requirement.

For a multiplier vector $\mathbf{v}$, we define the core $\overleftarrow{\mathbf{v}}$ of $\mathbf{v}$ as:

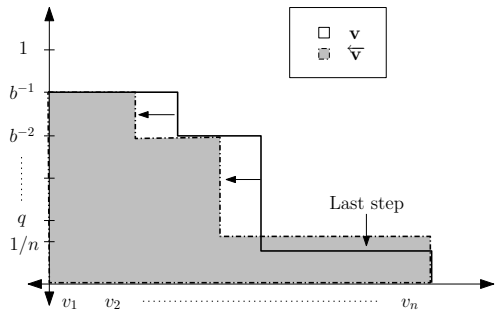


Figure 5: Horizontal fitting of $\mathbf{v}$.

Definition 4 (Core $\overleftarrow{\mathbf{v}}$). *The core $\overleftarrow{\mathbf{v}}$ of a multiplier vector $\mathbf{v}$ is the vector $\mathbf{u}$ constructed by Algorithm 2.*

Algorithm 2: ConstructCore

$i_1 \leftarrow 1$; $j_1 \leftarrow 1$;

while $i_1 \leq n$ **do**

$r \leftarrow \left(1 - \sum_{i=1}^{j_1-1} u_i\right) / (n - j_1 + 1)$;

if $v_{i_1} > r$ **then**

Find the largest index i_2 such that $v_{i_1} = v_{i_2}$;

Find largest integer k such that $\lceil a^k \rceil \leq (i_2 - j_1 + 1)$;

for $i = j_1$ *to* $j_1 + \lceil a^k \rceil - 1$ **do**
$u_i \leftarrow v_{i_1}$;

$i_1 \leftarrow i_2 + 1$;

$j_1 \leftarrow j_1 + \lceil a^k \rceil$;

else

for $i = j_1$ *to* n **do**
$u_i \leftarrow r$;

break

Operation of Algorithm 2: Each iteration of the `while` loop processes one step of $\mathbf{v}$ and $\mathbf{u}$. i_1 and j_1 hold the first index of the current step of $\mathbf{v}$ and $\mathbf{u}$ respectively. r is the minimum height of the current step of $\mathbf{u}$ by monotonicity. If $r \geq v_{i_1}$, then the requirement for unit l_1 norm forces us to introduce the last step of the staircase of $\mathbf{u}$. Otherwise, $[i_1, ..., i_2]$ is the current step of $\mathbf{v}$ and we set the width of the current step of $\mathbf{u}$ to be $\lceil a^k \rceil$.

Observation 6. *The core of a vector $\mathbf{v}$ is a multiplier vector itself, i.e. it has non-increasing*

components and unit l_1 norm. Moreover, $\mathbf{v}$ *dominates* $\overleftarrow{\mathbf{v}}$.

Proof. The algorithm to construct $\mathbf{u} = \overleftarrow{\mathbf{v}}$ itself easily ensures the unit norm. To prove monotonicity by contradiction, assume that there exists i such that $u_i < u_{i+1}$. This can only happen is $i+1$ is the first index of the last step of $\mathbf{u}$ and i is the last index of the penultimate step. Let j be the first index of the penultimate step. Then

$$1 - \sum_{k=1}^{j-1} u_k \;=\; \sum_{k=j}^{n} u_k \;=\; (i-j+1)u_i + (n-i)u_{i+1} \;>\; (n-j+1)\cdot u_i$$

This means

$$u_j \;=\; u_i \;<\; \left(1 - \sum_{k=1}^{j-1} u_k\right)/(n-j+1)$$

which is impossible since the right-hand side of the above inequality is the minimum value u_j could have been assigned.

To see that $\mathbf{v}$ dominates $\mathbf{u} = \overleftarrow{\mathbf{v}}$, observe that if the index i does not belong to the last step of $\mathbf{u}$, then $u_i = v_j$ for some $j \geq i$, and hence $u_i = v_j \leq v_i$. Now consider the smallest i such that $u_i > v_i$. Then i must belong to the last step of $\mathbf{u}$. Therefore, $u_j = u_i > v_i \geq v_j$ for all $j \geq i$. □

Lemma 18. *For any multiplier vector* $\mathbf{v}$ *and allocation* $\mathbf{S}$, $f(\overleftarrow{\mathbf{v}}, \mathbf{S}) \;\geq\; f(\mathbf{v}, \mathbf{S})/a$.

Proof. Suppose the staircase of $\mathbf{v}$ has s_1 steps and that of $\mathbf{u} = \overleftarrow{\mathbf{v}}$ has s_2 steps. Then the following four properties follow directly from the algorithm:

1. $s_2 \leq s_1$
2. For $1 \leq i < s_2$, the i'th step of $\mathbf{v}$ is at most a times as wide as the i'th step of $\mathbf{u}$ and both have the same height.
3. For $1 \leq i \leq s_2$, let i_1 and j_1 be the first indices of the i'th steps of $\mathbf{v}$ and $\mathbf{u}$ respectively. Then $i_1 \geq j_1$.
4. If $[j, ..., n]$ is the last step of $\mathbf{u}$ then $u_i \geq v_i$ for $i \geq j$.

To prove the lemma, we will compare the the contributions of corresponding steps of the staircases of $\mathbf{v}$ and $\mathbf{u}$ to the objective functions.

For $i < s_2$, let $[i_1, ..., i_2]$ be the i'th step of $\mathbf{v}$, $[j_1, ..., j_2]$ be the i'th step of $\mathbf{u}$ and $h = v_{i_1} = u_{j_1}$ be their common height. We have

$$\sum_{k=j_1}^{j_2} u_k f(S_k) \;=\; h \sum_{k=j_1}^{j_2} f(S_k) \;\geq\; h \sum_{k=i_1}^{i_i+j_2-j_1} f(S_k)$$

by the third property. The monotonicity of $\mathbf{S}$ and the second property then imply

$$\sum_{k=j_1}^{j_2} u_k f(S_k) \;\geq\; \frac{1}{a} \sum_{k=i_1}^{i_2} v_k f(S_k)$$

So the i'th step of $\mathbf{v}$ contributes at most a times value to $f(\mathbf{v}, \mathbf{S})$ as the i'th step of $\mathbf{u}$ contributes to $f(\mathbf{u}, \mathbf{S})$, where $i < s_2$.

Finally by the fourth property, the step s_2 of $\mathbf{u}$ contributes more to $f(\mathbf{u}, \mathbf{S})$ than the corresponding contribution of steps $s_2, ..., s_1$ of $\mathbf{v}$ to $f(\mathbf{v}, \mathbf{S})$ combined. The result therefore follows. □

We now define our set of vectors $\mathcal{U}$ as follows: $\mathcal{U} = \{ \overleftarrow{\mathbf{v}} : \mathbf{v} \in \mathcal{U}' \}$. We populate the range $\mathcal{R}$ of allocations as $\mathcal{R} = \{ \mathrm{B}(\mathbf{v}) : \mathbf{v} \in \mathcal{U} \}$ where $\mathrm{B}(\mathbf{v})$ is the α-approximate allocation returned by the black box algorithm.

3.5.2 Proof of Theorem 14

We run the following maximal-in-range mechanism: Given an input multiplier vector $\mathbf{v}$ we return the allocation $\mathbf{T} \in \mathcal{R}$ that maximizes $f(\mathbf{v}, \mathbf{T})$. We need to prove that $f(\mathbf{v}, \mathbf{T}) \geq \frac{3\alpha}{4ab} \cdot \mathrm{OPT}(\mathbf{v})$

Let $\mathbf{S} = \mathrm{A}(\mathbf{v})$ be the optimal allocation for $\mathbf{v}$ and $\overleftarrow{\lfloor \mathbf{v} \rfloor}$ be *the core of the floor* of $\mathbf{v}$. Combining Lemmas 17 and 18, we conclude that $f(\overleftarrow{\lfloor \mathbf{v} \rfloor}, \mathbf{S}) \geq \frac{3}{4ab} \cdot \mathrm{OPT}(\mathbf{v})$. Since $\overleftarrow{\lfloor \mathbf{v} \rfloor} \in \mathcal{U}$, there exists an allocation $\mathbf{X} \in \mathcal{R}$ such that

$$f(\overleftarrow{\lfloor \mathbf{v} \rfloor}, \mathbf{X}) \;\geq\; \alpha \cdot \mathrm{OPT}(\overleftarrow{\lfloor \mathbf{v} \rfloor}) \;\geq\; \alpha \cdot f(\overleftarrow{\lfloor \mathbf{v} \rfloor}, \mathbf{S}) \;\geq\; \frac{3\alpha}{4ab} \cdot \mathrm{OPT}(\mathbf{v}) \tag{17}$$

Since $\mathbf{v}$ dominates $\lfloor \mathbf{v} \rfloor$ which in turn dominates $\overleftarrow{\lfloor \mathbf{v} \rfloor}$ (Refer to Observation 5 and 6), application of Lemma 13 yields:

$$f(\mathbf{v}, \mathbf{X}) \;\geq\; f(\lfloor \mathbf{v} \rfloor, \mathbf{X}) \;\geq\; f(\overleftarrow{\lfloor \mathbf{v} \rfloor}, \mathbf{X}) \tag{18}$$

Using equations (17) and (18),

$$f(\mathbf{v}, \mathbf{T}) \;\geq\; f(\mathbf{v}, \mathbf{X}) \;\geq\; f(\overleftarrow{\lfloor \mathbf{v} \rfloor}, \mathbf{X}) \;\geq\; \frac{3\alpha}{4ab} \cdot \mathrm{OPT}(\mathbf{v})$$

The running time of the mechanism is established by Lemma 19, which finishes the proof of Theorem 14.

Lemma 19. $|\mathcal{R}| \;=\; O\left((\log_a n)^{\log_b n}\right)$

Proof. $|\mathcal{R}|$ is bounded by $|\mathcal{U}|$. $\mathcal{U}$ consists of only those vectors which are cores of floors of some multiplier vectors. We have seen that each step of the staircase of $\mathbf{v} \in \mathcal{U}$ except the last must be of width $w = \lceil a^k \rceil$ for some integer k. Moreover, there can be only $|Q| = O(\log_b n)$ such steps and at most one of each height. We have also remarked that specifying all but the last step of a staircase fixes it. Therefore there can be at most $O\left((\log_a n)^{\log_b n}\right)$ distinct staircases in $\mathcal{U}$. □

3.6 Discussion

As shown in [18, 12], it seems that designing a truthful mechanism with good approximation factor for maximizing social welfare is a difficult problem. In light of this, our work suggests an important research direction to pursue in combinatorial auctions- to divide the valuation function into a part which is common among all the agents and can be estimated by the auctioneer and a part which is unique and private to individual agents.

Also, it would be interesting to see if for submodular public functions (or even more specifically, for coverage functions), which have concrete motivation in TV ad auctions, one can design a constant factor polynomial time truthful mechanism.

CHAPTER IV

ONLINE VERTEX-WEIGHTED BIPARTITE MATCHING AND SINGLE-BID BUDGETED ALLOCATIONS

Online bipartite matching is a fundamental problem with numerous applications such as matching candidates to jobs or boys to girls. More recently, this and related problems have received significant attention, because they model the allocation aspect of sponsored search auctions, where multiple agents (advertisers) bid on items (query keywords) which arrive one by one in an online manner. In this chapter, we look at the following vertex-weighted version of this problem:

Online vertex-weighted bipartite matching: The input instance is a bipartite graph $G(U, V, E, \{b_u\}_{u \in U})$, with the vertices in U and their weights b_u known ahead of time. Vertices in V arrive one at a time, online, revealing their incident edges. An arriving vertex can be matched to an unmatched neighbor upon arrival. Matches once made cannot be revoked later and a vertex left unmatched upon arrival cannot be matched later. The goal is to maximize the sum of the weights of the matched vertices in U.

When all the weights are equal, this reduces to the classic *online bipartite matching* problem for which Karp, Vazirani and Vazirani gave an optimal $\left(1 - \frac{1}{e}\right)$-competitive algorithm in their seminal work [38].

Our main result is an optimal $\left(1 - \frac{1}{e}\right)$-competitive randomized algorithm for general vertex weights. Our solution constitutes the first known generalization of the algorithm in [38] in this model and provides new insights into the role of randomization in online allocation problems. It also effectively solves the problem of *online budgeted allocations* [47] in the case when an agent makes the same bid for any desired item, even if the bid is comparable to his budget - complementing the results of [47, 11] which apply when the bids

are much smaller than the budgets.

4.1 Motivation, Background and Overview of Our Result

Online bipartite matching is a fundamental problem with numerous applications such as matching candidates to jobs, ads to advertisers, or boys to girls. A canonical result in online bipartite matching is due to Karp, Vazirani and Vazirani [38], who gave an optimal online algorithm for the unweighted case to maximize the *size* of the matching. In their model, we are given a bipartite graph $G(U, V, E)$. The vertices in U are known ahead of time, while the vertices in V arrive one at a time online in an arbitrary order. When a vertex in V arrives, the edges incident to it are revealed and it can be matched to a neighboring vertex in U that has not already been matched. A match once made cannot be revoked. The goal is to maximize the number of matched vertices.

However, in many real world scenarios, the value received from matching a vertex might be different for different vertices: (1) Advertisers in online display ad-campaigns are willing to pay a fixed amount every time their graphic ad is shown on a website. By specifying their targeting criteria, they can choose the set of websites they are interested in. Each impression of an ad can be thought of as matching the impression to the advertiser, collecting revenue equal to the advertiser's bid. (2) Consider the sale of an inventory of items such as cars. Buyers arrive in an online manner looking to purchase one out of a specified set of items they are interested in. The sale of an item generates revenue equal to the price of the item. The goal in both these cases is to maximize the total revenue.

Connection to the online budgeted allocation problem: Apart from being a natural generalization of the online bipartite matching problem, our vertex-weighted matching problem is closely related to an important class of online problems. Mehta *et al* [47] considered the following online version of maximum budgeted allocation problem [24, 45] to model sponsored search auctions: We have n agents and m items. Each agent i specifies a monetary budget B_i and a bid b_{ij} for each item j. Items arrive online, and must be immediately allocated to an agent. If a set S of items is allocated to agent i, then the agent

pays the minimum of B_i and $\sum_{j \in S} b_{ij}$. The objective is to maximize the total revenue of the algorithm. An important and unsolved restricted case of this problem is when all the non-zero bids of an agent are equal, *i.e.* $b_{ij} = b_i$ or 0 for all j. This case reduces to our vertex-weighted matching problem (For a proof, refer to Section 4.5.3).

For the general online budgeted allocation problem, no factor better than $\frac{1}{2}$ is yet known. This factor is achieved by a simple deterministic greedy algorithm given by Lehmann *et al* [45]. The same algorithm also provides factor $\frac{1}{2}$ for online *submodular* allocation, where the agents have a general submodular valuation function over the set of items. For visual representation of the hierarchy of these online allocation problems, refer to Figure 6. The best known lower bound stands at $1 - \frac{1}{e}$ due to the hardness result in [38] for the case when all bids and budgets are equal to 1 - which is equivalent to the unweighted online matching problem. The *small bids* case - where $b_{ij} \ll B_i$ for all i and j - was solved by [47, 11] achieving the optimal $1 - \frac{1}{e}$ deterministic competitive ratio. It was believed that handling *large bids* requires the use of randomization, as in [38]. In particular, many attempts [43, 8, 27] had been made to simplify the analysis of the randomized algorithm in [38], but no generalization had been achieved.

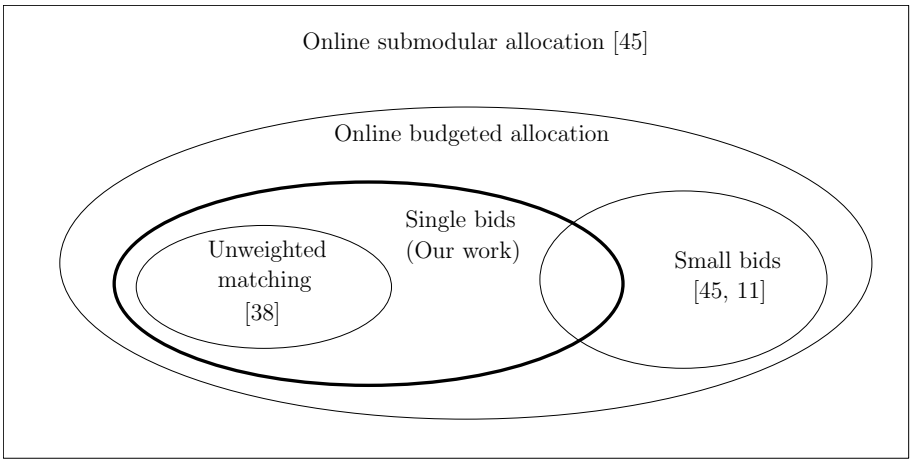


Figure 6: Hierarchy of related online allocation problems studied in literature.

Our solution to the vertex-weighted matching problem is a significant step in this direction. Our algorithm generalizes that of [38] and provides new insights into the role of

randomization in these solutions, as outlined in Section 4.1.1. Finally, our algorithm has interesting connections to the solution of [47] for the *small bids* case - despite the fact that the vertex-weighted matching problem is neither harder nor easier than the *small bids* case. This strongly suggests a possible unified approach to the unrestricted online budgeted allocation problem. See Section 4.5 for details.

4.1.1 Overview of the Result

Solution to the unweighted case: To describe our result, it is instructive to start at the unweighted case ($b_u = 1$ for all $u \in U$) and study its solution by [38]. Two natural approaches that match each arriving $v \in V$ to the an unmatched neighbor in U chosen (a) arbitrarily and (b) randomly, both fail to achieve competitive ratio better than $\frac{1}{2}$. Their solution is an elegant randomized algorithm called RANKING that works as follows: it begins by picking a *uniformly random permutation* of the vertices in U (called the "ranking" of the vertices). Then, as a vertex in V arrives, it is matched to the highest-ranked unmatched neighbor. Surprisingly, this idea of using correlated randomness for all the arriving vertices achieves the optimal competitive ratio of $1 - \frac{1}{e}$.

How do we generalize RANKING in presence of unrestricted weights b_u? The natural GREEDY algorithm which matches an arriving vertex to the highest-weighted unmatched neighbor, achieves a competitive ratio of $\frac{1}{2}$ (see Appendix B.1 for a proof). No deterministic algorithm can do better. While the optimality of RANKING for unweighted matching suggests choosing random ranking permutations of U, RANKING itself can do as badly as factor $\frac{1}{n}$ for some weighted instances.

The main challenge in solving this problem is that a good algorithm must follow very different strategies depending on the weights in the input instance. GREEDY and RANKING are both suboptimal for this problem, but both have ideas which are essential to its solution. In particular, they perform well on distinct classes of inputs, namely, GREEDY on highly skewed weights and RANKING on equal weights. The following observation about RANKING helps us bridge the gap between these two approaches: Suppose we perturb each

weight b_u identically and independently and then sort the vertices in the order of decreasing perturbed weights. When all the weights are equal, the resulting order happens to be a uniformly random permutation of U and thus, RANKING on unweighted instances can be thought of as GREEDY on perturbed weights! We use this insight to construct our solution to the vertex-weighted matching problem. While the nature of perturbation used did not matter in the above discussion, we need a very specific perturbation procedure for general vertex-weights.

Our algorithm is defined below:

Algorithm 3: PERTURBED-GREEDY

For each $u \in U$, pick a number x_u uniformly at random from $[0, 1]$.

Define the function $\psi(x) := 1 - e^{-(1-x)}$.

foreach *arriving* $v \in V$ **do**

Match v to the unmatched neighbor $u \in U$ with the highest value of $b_u\psi(x_u)$.

Break ties consistently, say by vertex id.

Remarks:

- It is not obvious, and indeed is remarkable in our opinion, that it suffices to perturb each weight b_u completely independently of other weights. The fact that PERTURBED-GREEDY achieves the best possible competitive ratio is a post-facto proof that such independence in perturbations is sufficient. Without the knowledge of our algorithm, one could reasonably believe that the vector of vertex-weights $\{b_u\}_{u\in U}$ - which is known offline - contains valuable information which can be exploited. In what follows we provide intuition as to why this is not the case.

  Consider the two input instances in Figure 7. Both the connected components in G_1 have equal weights, and hence we know that RANKING achieves the best possible competitive ratio on G_1. Similarly, both connected components in G_2 have highly skewed weights, suggesting GREEDY as the optimal algorithm. On the other hand, RANKING and GREEDY are far from optimal on G_2 and G_1 respectively. Since two instances with identical values of vertex-weights require widely differing strategies, this exercise suggests that we may not be losing must information by perturbing weights

independently. The optimality of our algorithm proves this suggestion.

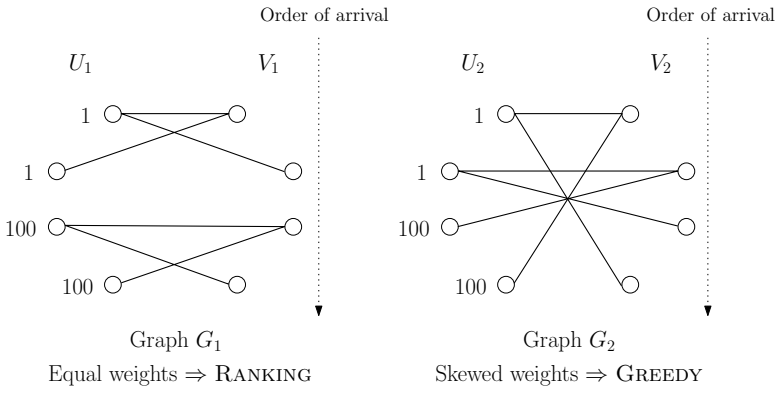


Figure 7: Two instances with the same vertex-weights, but widely differing optimal strategies.

- The particular form of the function ψ is not a pre-conceived choice, but rather an artifact of our analysis. This combined with the discussion in Section 4.5 seems to suggest that ψ is the 'right' perturbation function.
- We note that we can also choose the function $\psi(x)$ to be $1 - e^{-x}$, which keeps the algorithm and results unchanged.
- Finally, we note that the multipliers $y_u = \psi(x_u)$ are distributed according to the density function $f(y) = \frac{1}{1-y}$ for $y \in \left[0, 1 - \frac{1}{e}\right]$. Therefore, we could have equivalently stated our algorithm as: For each $u \in U$, choose a random multiplier $y_u \in \left[0, 1 - \frac{1}{e}\right]$ from the above distribution, and use $b_u y_u$ as the perturbed weight.

Our main result is the following theorem. The second part of the theorem follows from the optimality of RANKING for unweighted matching [38].

Theorem 20. PERTURBED-GREEDY *achieves a competitive ratio of* $1 - 1/e$ *for the vertex-weighted online bipartite matching problem. No (randomized) algorithm has a better competitive ratio.*

In addition to the basic idea (from the proof of RANKING) of charging unmatched vertices in some probabilistic events to matched vertices in other events, our analysis needs to handle

the new complexity introduced due to the weights on vertices. At a very high level, just like the algorithm, our analysis also manages to pull together the essence of the analyses of both Greedy and Ranking.

4.1.2 Related Work

Our problem is a special case of online bipartite matching with edge weights, which has been studied extensively in the literature. With general edge weights and vertices arriving in adversarial order, every algorithm can be arbitrarily bad (see Appendix B.2). There are two ways to get around this hardness: (a) assume that vertices arrive in a random order, and/or (b) assume some restriction on the edge weights.

When the vertices arrive in random order, it corresponds to a generalization of the *secretary* problem to transversal matroids [3]. Dimitrov and Plaxton [15] study a special case where the weight of an edge (u, v) depends only on the vertex v – this is similar to the problem we study, except that it assumes a random arrival model (and assumes vertex weights on the *online* side). Korula and Pal [42] give an $\frac{1}{8}$-competitive algorithm for the problem with general edge weights and for the general *secretary* problem on transversal matroids.

If one does not assume random arrival order, every algorithm can be arbitrarily bad with general edge weights or even with weights on arriving vertices. [37] introduce the assumption of edge weights coming from a metric space and give an optimal deterministic algorithm with a competitive factor of $\frac{1}{3}$. As far as we know, no better randomized algorithm is known for this problem.

Finally, there has been other recent work [14, 27, 23], although not directly related to our results, which study online bipartite matching and budgeted allocations in stochastic arrival settings.

Roadmap: The rest of this section is structured as follows: In Section 4.2 we set up the preliminaries and provide a warm up analysis of a proof of Ranking in the unweighted special case. Section 4.3 contains the proof of Theorem 20.

4.2 Preliminaries

4.2.1 Problem Statement

Consider an undirected bipartite graph $G(U, V, E)$. The vertices of U, which we will refer to as the *offline* side, are known from the start. We are also given a weight b_u for each vertex $u \in U$. The vertices of V, referred to as the *online* side, arrive one at a time (in an arbitrary order). When a vertex v arrives, all the edges incident to it are revealed, and at this point, the vertex v can be matched to one of its unmatched neighbors (irrevocably) or left permanently unmatched. The goal is to maximize the sum of the weights of matched vertices in U.

Let permutation π represent the arrival order of vertices in V and let M be the subset of matched vertices of U at the end. Then for the input (G, π), the gain of the algorithm, denoted by $ALG(G, \pi)$, is $\sum_{u \in M} b_u$.

We use competitive analysis to analyze the performance of an algorithm. Let $M^*(G)$ be an optimal (offline) matching, i.e. one that maximizes the total gain for G (note that the optimal matching depends only on G, and is independent of π), and let $\mathrm{OPT}(G)$ be the total gain achieved by $M^*(G)$. Then the competitive ratio of an algorithm is $\min_{G,\pi} \frac{\mathrm{ALG}(G,\pi)}{\mathrm{OPT}(G)}$. Our goal is to devise an online algorithm with a high competitive ratio.

Definition 5 ($M^*(G)$). *For a given G, we will fix a particular optimal matching, and refer to it as the optimal offline matching $M^*(G)$.*

Definition 6 (u^*). *Given a G, its optimal offline matching $M^*(G)$ and a $u \in U$ that is matched in $M^*(G)$, we define $u^* \in V$ as its partner in $M^*(G)$.*

4.2.2 Warm-up: Analysis of RANKING for Unweighted Online Bipartite Matching

Recall that online bipartite matching is a special case of our problem in which the weight of each vertex is 1, i.e. $b_u = 1$ for all $u \in U$. [38] gave an elegant randomized algorithm for this problem and showed that it achieves a competitive ratio of $(1 - 1/e)$ in expectation. In this section, we will re-prove this classical result as a warm-up for the proof of the main result. The following proof is based on those presented by [8, 27] previously.

Algorithm 4: RANKING

Choose a random permutation σ of U uniformly from the space of all permutations.

foreach *arriving* $v \in V$ **do**

Match v to the unmatched neighbor in u which appears earliest in σ.

Theorem 21 ([38]). *In expectation, the competitive ratio of* RANKING *is at least* $1 - \frac{1}{e}$.

In this warm-up exercise, we will simplify the analysis by making the following assumptions: $|U| = |V| = n$ and G has a perfect matching. These two assumptions imply that $\text{OPT} = n$ and that the optimal matching $M^*(G)$ is a perfect matching.

For any permutation σ, let RANKING(σ) denote the matching produced by RANKING when the randomly chosen permutation happens to be σ. For a permutation $\sigma = (u_1, u_2, ..., u_n)$ of U, we say that a vertex $u = u_t$ has rank $\sigma(u) = t$. Consider the random variable

$$y_{\sigma,i} = \begin{cases} 1 & \text{If the vertex at rank } i \text{ in } \sigma \text{ is matched by RANKING}(\sigma). \\ 0 & \text{Otherwise} \end{cases}$$

Definition 7 (Q_t, R_t). *Q_t is defined as the set of all occurrences of matched vertices in the probability space.*

$$Q_t = \{\ (\sigma, t)\ :\ y_{\sigma,t} = 1\ \}$$

Similarly, R_t is defined as the set of all occurrences of unmatched vertices in the probability space.

$$R_t = \{\ (\sigma, t)\ :\ y_{\sigma,t} = 0\ \}$$

Let x_t be the probability that the vertex at rank t in σ is matched in RANKING(σ), over the random choice of permutation σ. Then, $x_t = \frac{|Q_t|}{n!}$ and $1 - x_t = \frac{|R_t|}{n!}$. The expected gain of the algorithm is $\text{ALG}_{G,\pi} = \sum_t x_t$.

Definition 8 (σ_u^i). *For any σ, let σ_u^i be the permutation obtained by removing u from σ and inserting it back into σ at position i.*

Lemma 22. *If the vertex u at rank t in σ is unmatched by* RANKING*(σ), then for every $1 \le i \le n$, u^* is matched in* RANKING*(σ_u^i) to a vertex u' such that $\sigma_u^i(u') \le t$.*

Proof. Refer to Lemma 24 in the analysis of PERTURBED-GREEDY for the proof of a more general version of this statement. □

In other words, for every vertex that remains unmatched in some event in the probability space, there are many matched vertices in many different events in the space. In the remaining part of this section, we quantify this effect by bounding $1 - x_t$, which is the probability that the vertex at rank t in σ (chosen randomly by RANKING) is unmatched, in terms of some of the x_ts.

Definition 9 (Charging map $f(\sigma, t)$). *f is a map from bad events (where vertices remain unmatched) to good events (where vertices get matched). For each $(\sigma, t) \in R_t$,*

$$f(\sigma,t) \;=\; \{(\sigma_u^i, s) \;:\; 1 \le i \le n,\; \sigma(u) = t \text{ and } \textsc{Ranking}(\sigma_u^i) \text{ matches } u^* \text{ to } u' \text{ where } \sigma_u^i(u') = s\}$$

In other words, let u be the vertex at rank t in σ. Then $f(\sigma, t)$ contains all (σ', s), such that σ' can be obtained from σ by moving u to some position and s is the rank of the vertex to which u^*, the optimal partner of u, is matched in σ'.

For every $(\sigma, t) \in R_t$, $(\pi, s) \in f(\sigma, t)$ implies $y_{\pi,s} = 1$ for some $s \le t$. Therefore,

$$\bigcup_{(\sigma,t)\in R_t} f(\sigma,t) \;\subseteq\; \bigcup_{s\le t} Q_s$$

Claim 23. *If $(\rho, s) \in f(\sigma, t)$ and $(\rho, s) \in f(\overline{\sigma}, t)$, then $\sigma = \overline{\sigma}$.*

Proof. Let u' be the vertex in ρ at rank s. Let u^* be the vertex to which u' is matched by RANKING. Then it is clear from the definition of the map f that $\rho = \sigma_u^{\rho(u)} = \overline{\sigma}_u^{\rho(u)}$, implying $\sigma = \overline{\sigma}$. □

The claim proves that for a fixed t, the set-values $f(\sigma, t)$ are disjoint for different σ. Therefore,

$$1 - x_t \;=\; \frac{|R_t|}{n!} \;=\; \frac{1}{n} \cdot \frac{\left|\bigcup_{(\sigma,t)\in R_t} f(\sigma,t)\right|}{n!} \;\le\; \frac{1}{n} \cdot \frac{\left|\bigcup_{s\le t} Q_s\right|}{n!} \;=\; \frac{1}{n} \sum_{s\le t} \frac{|Q_s|}{n!} \;=\; \frac{\sum_{s\le t} x_s}{n}$$

Therefore, the probabilities x_t's obey the equation $1 - x_t \;\le\; \frac{1}{n}\sum_{s\le t} x_s$ for all t. Since any vertex with rank 1 in any of the random permutations will be matched, $x_1 = 1$. One can make simple arguments [38, 8, 27] to prove that under these conditions, $\text{ALG}_{G,\pi} = \sum_t x_t \ge \left(1 - \frac{1}{e}\right) n = \left(1 - \frac{1}{e}\right) OPT$, thereby proving Theorem 21.

4.3 Proof Of Theorem 20

In this section, we will assume that $|U| = |V| = n$ and that G has a perfect matching. In Appendix 4.4 we will show how this assumption can be removed.

Recall that our algorithm works as follows: For each $u \in U$, let $\sigma(u)$ be a number picked uniformly at random from $[0, 1]$ (and independent of other vertices) Now, when the next vertex $v \in V$ arrives, match it to the available neighbor u with the maximum value of $b_u\psi(\sigma(u))$, where $\psi(x) := 1 - e^{-(1-x)}$.

For ease of exposition, we will prove our result for a discrete version of this algorithm. For every $u \in U$ we will choose a random integer $\sigma(u)$ uniformly from $\{1, ..., k\}$ where k is the parameter of discretization. We will also replace the function $\psi(x)$ by its discrete version $\psi(i) = 1 - \left(1 - \frac{1}{k}\right)^{-(k-i+1)}$. The discrete version of our algorithm also matches each incoming vertex $v \in V$ to the available neighbor u with the maximum value of $b_u\psi(\sigma(u))$. Notice that ψ is a decreasing function, so $\psi(s) \geq \psi(t)$ if $s \leq t$. As $k \to \infty$, the discrete version tends to our original algorithm.

We begin with some definitions, followed by an overview of the proof.

We will denote by $\sigma \in [k]^n$, the set of these random choices. We will say that u is at *position* t in σ if $\sigma(u) = t$. As a matter of notation, we will say that position s is *lower* (resp. higher) than t if $s \leq t$ (resp. $s \geq t$).

Definition 10 (u is matched in σ). *We say that u is matched in σ if our algorithm matches it when the overall choice of random positions happens to be σ.*

Definition 11 (Q_t, R_t). *Q_t is defined as the set of all occurrences of matched vertices in the probability space.*

$$Q_t = \{(\sigma, t, u) \ : \ \sigma(u) = t \text{ and } u \text{ is matched in } \sigma\}$$

Similarly, R_t is defined as the set of all occurrences of unmatched vertices in the probability space.

$$R_t = \{(\sigma, t, u) \ : \ \sigma(u) = t \text{ and } u \text{ is not matched in } \sigma\}$$

Let x_t be the *expected gain* at t, over the random choice of σ. Then,

$$x_t = \frac{\sum_{(\sigma,t,u)\in Q_t} b_u}{k^n} \tag{19}$$

The expected gain of the algorithm is $\text{ALG}_{G,\pi} = \sum_t x_t$. Also note that the *optimal gain* at any position t is $B = \frac{\text{OPT}(G)}{k}$ since each vertex in U appears at position t with probability $1/k$ and is matched in the optimal matching. Therefore,

$$B - x_t = \frac{\sum_{(\sigma,t,u)\in R_t} b_u}{k^n} \tag{20}$$

Definition 12 (σ_u^i). *For any σ, $\sigma_u^i \in [k]^n$ is obtained from σ by changing the position of u to i, i.e. $\sigma_u^i(u) = i$ and $\sigma_u^i(u') = \sigma(u')$ for all $u' \neq u$.*

Observation 7. *For all $(\sigma, t, u) \in R_t$ and $1 \leq i \leq k$, our algorithm matches u^* to some $u' \in U$ in σ_u^i.*

The above observation follows from Lemma 24. We'll use it to define a map from bad events to good events as follows.

Definition 13 (Charging Map $f(\sigma, t, u)$). *For every $(\sigma, t, u) \in R_t$, define the set-valued map*

$$f(\sigma,t,u) = \{(\sigma_u^i, s, u') \ : \ 1 \leq i \leq k, \text{ and the algorithm matches } u^* \text{ to } u' \text{ in } \sigma_u^i \text{ where } \sigma_u^i(u') = s\}$$

Observation 8. *If $(\rho, s, u') \in f(\sigma, t, u)$, then $(\rho, s, u') \in Q_s$.*

Now we are ready to give an overview of the proof.

4.3.1 Overview of the proof

The key idea in the analysis of RANKING in Section 4.2.2 was that we can bound the number of occurrences of unmatched vertices - the *bad* events - in the entire probability space by a careful count of the matched vertices - the *good* events. The charging map f defined above is an attempt to do this. We'll show in Lemma 24 that if $(\sigma_u^i, s, u') \in f(\sigma, t, u)$, then the scaled (by ψ) gain due to u' in σ_u^i is no less than the scaled loss due to u in σ. However, s may be higher or lower than t, unlike RANKING where $s \leq t$. This implies that the bound

is in terms of events in $\bigcup_s Q_s$, $1 \leq s \leq k$, which is very weak (as many of the events in the union are not used).

One idea is to bound the sum of losses incurred at all positions, thereby using almost all the events in $\bigcup_s Q_s$. However, if we do this, then the charging map loses the disjointness property, i.e. if $(\sigma, t, u) \in R_t$ and $(\sigma_u^i, i, u) \in R_i$ then f value of both these occurrences is the same. Thus, each event in $\bigcup_s Q_s$ gets charged several times (in fact a non-uniform number of times), again making the bound weak. To this end, we introduce the idea of *marginal loss* (Definition 14), which helps us define a disjoint map and get a tight bound.

Next, we formalize the above.

4.3.2 Formal proof

We begin by proving an analogue of Lemma 22.

Lemma 24. *If the vertex u at position t in σ is unmatched by our algorithm, then for every $1 \leq i \leq k$, the algorithm matches u^* in σ_u^i to a vertex u' such that $\psi(t)b_u \leq \psi\left(\sigma_u^i(u')\right) b_{u'}$.*

Proof. **Case 1 ($i \geq t$)**: Let $v_1, ..., v_n$ be the order of arrival of vertices in V. Clearly, v_1 will see the same choice of neighbors in σ_u^i as in σ, except the fact that the position of u is higher in σ_u^i than in σ. Since we did not match v_1 to u in σ, v_1 will retain its match from σ even in σ_u^i. Now assuming that $v_1, ..., v_l$ all match the same vertex in σ_u^i as they did in σ, v_{l+1} will see the same choice of neighbors in σ_u^i as in σ with the exception of u. Since v_{l+1} did not match u in σ either, it will retain the same neighbor in σ_u^i and by induction every vertex from V, specifically u^* keeps the same match in σ_u^i as in σ. Since $\sigma(u') = \sigma_u^i(u')$, we conclude $\psi(t)b_u \leq \psi\left(\sigma_u^i(u')\right) b_{u'}$.

Case 2 ($i < t$): For a vertex $v \in V$, let $m_\sigma(v)$ and $m_{\sigma_u^i}(v)$ be the vertices to which v is matched in σ and σ_u^i respectively, if such a match exists and null otherwise. Intuitively, since $\psi(i) \geq \psi(t)$, the scaling factor of b_u only improves in this case, while that of any other vertex in U remains the same. Therefore, we can expect u to be more likely to be matched in σ_u^i and the $\psi\left(\sigma_u^i\left(m_{\sigma_u^i}(v)\right)\right) b_{m_{\sigma_u^i}(v)} \geq \psi\left(\sigma\left(m_\sigma(v)\right)\right) b_{m_\sigma(v)}$ to hold for all $v \in V$. In fact, something more specific is true. The symmetric difference of the two matchings

produced by the algorithm for σ and σ_u^i is exactly one path starting at u that looks like $(u,\ v_1,\ m_\sigma(v_1),\ v_2,\ m_\sigma(v_2),\ ...)$, where $(v_1, v_2, ...)$ appear in their order of arrival. In what follows we prove this formally.

Let $V' = \{v \in V\ :\ m_\sigma(v) \neq m_{\sigma_u^i}(v)\}$ be the set of vertices in V with different matches in σ and σ_u^i. Index the members of V' as $v_1, ..., v_l$ in the same order as their arrival, *i.e.* v_1 arrives the earliest. For simplicity, let $u_j = m_\sigma(v_j)$ and $w_j = m_{\sigma_u^i}(v_j)$.

We assert that the following invariant holds for $2 \leq j \leq l$: Both u_j and u_{j-1} are unmatched in σ_u^i when v_j arrives and v_j matches u_{j-1}, *i.e.* $w_j = u_{j-1}$.

For base case, observe that the choice of neighbors for v_1 in σ_u^i is the same as in σ, except u, which has moved to a lower position. Since by definition v_1 does not match u_1 in σ_u^i, $w_1 = u$. Now consider the situation when v_2 arrives. All the vertices arriving before v_2 - with the exception of v_1 - have been matched to the same vertex in σ_u^i as in σ, and v_1 has matched to u, leaving u_1 yet unmatched. Let $U_\sigma(v_2)$ and $U_{\sigma_u^i}(v_2)$ be the sets of unmatched neighbors of v_2 in σ and σ_u^i respectively *at the moment* when v_2 arrives. Then from above arguments, $U_{\sigma_u^i}(v_2) = (U_\sigma(v_2) \cup \{u_1\}) - \{u\}$. Since u was unmatched in σ, $u_2 \neq u$. Since $v_2 \in V'$, $w_2 \neq u_2$. This is only possible if $w_2 = u_1$. And hence the base case is true.

Now assume that the statement holds for $j-1$ and consider the arrival of v_j. By induction hypothesis, v_1 has been matched to u and $v_2, .., v_{j-1}$ have been matched to $u_1, ..., u_{j-2}$ respectively. All the other vertices arriving before v_j that are not in V' have been matched to the same vertex in σ_u^i as in σ. Therefore, u_{j-1} is yet unmatched. Let $U_\sigma(v_j)$ and $U_{\sigma_u^i}(v_j)$ be the sets of unmatched neighbors of v_j in σ and σ_u^i respectively at the moment when v_j arrives. Then from above arguments, $U_{\sigma_u^i}(v_j) = (U_\sigma(v_j) \cup \{u_{j-1}\}) - \{u\}$. Since u was unmatched in σ, $u_j \neq u$. Given that $w_j \neq u_j$, the only possibility is $w_j = u_{j-1}$. Hence the proof of the inductive statement is complete.

If $u^* \notin V'$ then $u' = m_{\sigma_u^i}(u^*) = m_\sigma(u^*)$ and the statement of the lemma clearly holds since $\sigma(u') = \sigma_u^i(u')$. If $u^* = v_1$, then $u' = u$ and $\psi\left(\sigma_u^i(u')\right) b_{u'} = \psi(i) b_u \geq \psi(t) b_u$ since

$i < t$. Now suppose $u^* = v_j$ for some $j \geq 2$. Then $u' = u_{j-1}$ and by the invariant above,

$$\psi\left(\sigma_u^i(u')\right) b_{u'} \;=\; \psi\left(\sigma_u^i(u_{j-1})\right) b_{u_{j-1}} \;\geq\; \psi\left(\sigma_u^i(u_j)\right) b_{u_j} \tag{21}$$

$$= \;\psi\left(\sigma(u_j)\right) b_{u_j} \tag{22}$$

$$\geq \;\psi(t) b_u \tag{23}$$

Equation (21) follows from the fact that $u^* = v_j$ was matched in σ_u^i to u_{j-1} when u_j was also unmatched. The fact that only u changes its position between σ and σ_u^i leads us to (22). Finally, equation (23) follows from the fact that u^* was matched to u_j in σ when u was also unmatched. □

Using the above lemma, we get the following easy observation.

Observation 9. *For all $(\sigma, t, u) \in R_t$, $1 \leq t \leq k$, $f(\sigma, t, u)$ contains k values.*

Remark: As noted in the overview, although Lemma 24 looks very similar to Lemma 22, it is not sufficient to get the result, since the good events pointed to by Lemma 24 are scattered among all positions $1 \leq s \leq k$ – in contrast to Lemma 22, which pointed to only lower positions $s \leq t$, giving too weak a bound. We try to fix this by combining the losses from all R_t. However we run into another difficulty in doing so. While for any fixed t, the maps $f(\sigma, t, u)$ are disjoint for all $(\sigma, t, u) \in R_t$, but the maps for two occurrences in different R_ts may not be disjoint. In fact, whenever some u is unmatched in σ at position t, it will also remain unmatched in σ_u^j for $j > t$, and the sets $f(\sigma, t, u)$ and $f(\sigma_u^j, j, u)$ will be exactly the same! This situation is depicted in Figure 8.

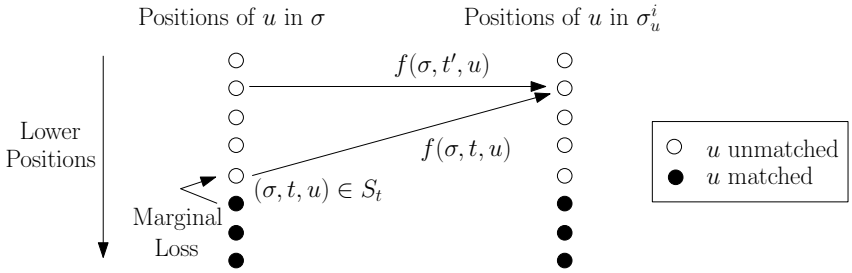


Figure 8: Marginal Losses

This absence of disjointness again renders the bound too weak. To fix this, we carefully select a subset of bad events from $\bigcup_t R_t$ such that their set-functions are indeed disjoint, while at the same time, the total gain/loss can be easily expressed in terms of the bad events in this subset.

Definition 14 (Marginal loss events S_t). *For $t > 1$, $S_t = \{(\sigma, t, u) \in R_t : (\sigma_u^{t-1}, t-1, u) \notin R_{t-1}\}$.*

Informally, S_t consists of *marginal* losses. If u is unmatched at position t in σ, but matched at position $t-1$ in σ_u^{t-1}, then $(\sigma, t, u) \in S_t$ (See Figure 8). The following property can be proved using the same arguments as in Case 1 in the proof of Lemma 24.

Observation 10. *For $(\sigma, t, u) \in S_t$, u is matched at i in σ_u^i if and only if $i < t$.*

Definition 15 (Expected Marginal Loss α_t).

$$\textit{Expected marginal loss at position } t \;=\; \alpha_t \;=\; \frac{\sum_{(\sigma,t,u)\in S_t} b_u}{k^n} \tag{24}$$

Claim 25.

$$\forall\ t: \qquad x_t \;=\; B - \sum_{s \le t} \alpha_s \tag{25}$$

$$\textit{Total loss} \;=\; \sum_t (B - x_t) \;=\; \sum_t (k - t + 1)\alpha_t \tag{26}$$

Proof. To prove equation (25), we will fix a t and construct a one-to-one map $g : R_t \to \bigcup_{s \le t} S_t$. Given $(\sigma, t, u) \in R_t$, let i be the lowest position of u such that u remains unmatched in σ_u^i. By observation 10, i is unique for (σ, t, u). We let $g(\sigma, t, u) = (\sigma_u^i, i, u)$. Clearly, $(\sigma_u^i, i, u) \in S_i$. To prove that the map is one-to-one, suppose $(\rho, s, u) = g(\sigma, t, u) = g(\overline{\sigma}, t, u)$. Then by definition of g, $\rho = \sigma_u^s = \overline{\sigma}_u^s$ which is only possible if $\sigma = \overline{\sigma}$. Therefore, $|R_t| = \bigcup_{s \le t} S_t$.

Lastly, observe that g maps an element of R_t corresponding to the vertex u being unmatched, to an element of S_i corresponding to the same vertex u being unmatched. From equation (20),

$$B - x_t \;=\; \frac{\sum_{(\sigma,t,u)\in R_t} b_u}{k^n} \;=\; \sum_{i \le t} \frac{\sum_{(\sigma_u^i,i,u)\in S_i} b_u}{k^n} \;=\; \sum_{i \le t} \alpha_i$$

This proves equation (25). Summing (25) for all t, we get (26). □

Now consider the same set-valued map f from Definition 13, but restricted only to the members of $\bigcup_t S_t$. We have:

Claim 26. *For $(\sigma, t, u) \in S_t$ and $(\overline{\sigma}, \overline{t}, \overline{u}) \in S_{\overline{t}}$, if $(\rho, s, u') \in f(\sigma, t, u)$ and $(\rho, s, u') \in f(\overline{\sigma}, \overline{t}, \overline{u})$ then $\sigma = \overline{\sigma}$, $t = \overline{t}$ and $u = \overline{u}$.*

Proof. If u' is matched to v in ρ then by definition of f, $v = u^* = \overline{u}^*$, implying $u = \overline{u}$. Therefore, $\rho = \sigma_u^i = \overline{\sigma}_u^i$ for some i. But this implies that $\overline{\sigma} = \sigma_u^j$ for some j. This is only possible for $j = t$ since by definition, if u is unmatched in σ at t, then there exists a unique i for which $(\sigma_u^i, i, u) \in \bigcup_t S_t$. If $j = t$, then $\sigma = \overline{\sigma}$ and $t = \overline{t}$. □

Armed with this disjointness property, we can now prove our main theorem.

Theorem 27. *As $k \to \infty$,*

$$\sum_t x_t \;\geq\; \left(1 - \frac{1}{e}\right) \mathrm{OPT}(G) \tag{27}$$

Proof. Using Lemma 24 and Observation 9, we have for every $(\sigma, t, u) \in S_t$,

$$\psi(t) b_u \;\leq\; \frac{1}{k} \sum_{(\sigma_u^i, s, u') \in f(\sigma,t,u)} \psi(s) b_{u'} \tag{28}$$

If we add the equation (28) for all $(\sigma, t, u) \in S_t$ and for all $1 \leq t \leq n$, then using Claim 26 and Observation 8, we arrive at

$$\begin{aligned}
\sum_t \psi(t) \frac{\sum_{(\sigma,t,u) \in S_t} b_u}{k^n} &\leq \frac{1}{k} \sum_t \psi(t) \frac{\sum_{(\sigma,t,u) \in Q_t} b_u}{k^n} \\
\sum_t \psi(t) \alpha_t &\leq \frac{1}{k} \sum_t \psi(t) x_t \qquad (29) \\
&= \frac{1}{k} \sum_t \psi(t) \left(B - \sum_{s \leq t} \alpha_s \right) \qquad (30)
\end{aligned}$$

Equation (29) follows from (24) and (19). Equation (30) uses Claim 25.

We now rearrange terms to get

$$\sum_t \alpha_t \left(\psi(t) + \frac{\sum_{s \geq t} \psi(s)}{k} \right) \;\leq\; \frac{B}{k} \sum_t \psi(t) \tag{31}$$

When $\psi(t) = 1 - \left(1 - \frac{1}{k}\right)^{k-t+1}$, observe that $\psi(t) + \frac{\sum_{s \geq t} \psi(s)}{k} \geq \frac{(k-t+1)}{k}$ and $\sum_t \psi(t) = \frac{k}{e}$ as $k \to \infty$. Using Claim 25,

$$\begin{aligned}
\text{Total loss} &= \sum_t (B - x_t) \;=\; \sum_t (k - t + 1)\alpha_t \\
&= k \sum_t \alpha_t \left(\psi(t) + \frac{\sum_{s \geq t} \psi(s)}{k} \right) \\
&\leq B \sum_t \psi(t) \\
&= \frac{kB}{e} \qquad\qquad \text{as } k \to \infty \\
&= \frac{\text{OPT}(G)}{e}
\end{aligned}$$

Hence, as $k \to \infty$,

$$\text{Total gain} \;\geq\; \left(1 - \frac{1}{e}\right) \text{OPT}(G)$$

Remark: Observe that we substituted for $\psi(t)$ only after equation (31) - up until that point, any choice of a non-increasing function ψ would have carried the analysis through. In fact, the chosen form of ψ is a result of trying to reduce the left hand side of equation (31) to the expected total loss. To conclude, the 'right' perturbation function is dictated by the analysis and not vice versa.

□

4.4 Graphs with Imperfect Matchings

In Section 4.3, we proved Theorem 20 for graphs $G(U, V, E)$ such that $|U| = |V|$ and G has a perfect matching. We can remove these assumptions with just a few modifications to the definitions and equations involved in the proof. The algorithm remains unchanged, *i.e.* we just use Perturbed-Greedy. We will only outline these modifications and the rest of the proof follows easily. Let $M^*(G)$ be a maximum weight matching in $G(U, V, E)$ and $\overline{U}$ be the set of vertices in U matched by $M^*(G)$. Thus we know that $\text{OPT}(G) = \sum_{u \in \overline{U}} b_u$.

Keeping the definition of Q_t the same, we change the definition of R_t to:

$$R_t = \{(\sigma, t, u) \;:\; u \in \overline{U} \text{ and } \sigma(u) = t \text{ and } u \text{ is not matched in } \sigma\}$$

The above redefinition conveys the fact that if a vertex u is *not* matched by $M^*(G)$, then we no longer consider u being unmatched a bad event. Consequently, equation (20) changes to:

$$B - x_t \leq \frac{\sum_{(\sigma,t,u)\in R_t} b_u}{k^n}$$

which in turn yields following counterpart of equation (25):

$$\forall t, \qquad x_t \;\geq\; B - \sum_{s\leq t} \alpha_s \tag{32}$$

Let Eq(t) be the version of (32) for t. We then multiply Eq(t) by $\psi(t) - \psi(t+1)$ and sum over $1 \leq t \leq n$ to obtain a combined inequality (with $\psi(k+1) = 0$):

$$\begin{aligned} \sum_t \left(\psi(t) \;-\; \psi(t+1)\right) x_t \;&\geq\; \psi(1)B \;-\; \sum_t \psi(t)\alpha_t \\ \sum_t \psi(t)\alpha_t \;&\geq\; \psi(1)\frac{\mathrm{OPT}(G)}{k} \;-\; \sum_t \frac{(1-\psi(t+1))}{k} x_t \end{aligned} \tag{33}$$

Equation (33) used the definition of $\psi(t) = 1 - \left(1 - \frac{1}{k}\right)^{(k-t+1)}$. Combining equation (33) with (29), we get:

$$\begin{aligned} \frac{1}{k}\sum_t \psi(t) x_t \;&\geq\; \psi(1)\frac{\mathrm{OPT}(G)}{k} - \sum_t \frac{(1-\psi(t+1))}{k} x_t \\ \sum_t x_t \;&\geq\; \psi(1)\mathrm{OPT}(G) - \sum_t \left(\psi(t) - \psi(t+1)\right) x_t \\ &\geq\; \left(1 - \frac{1}{e}\right)\mathrm{OPT}(G) \end{aligned}$$

as $k \to \infty$, since $\psi(1) \to \left(1 - \frac{1}{e}\right)$ and $\psi(t) - \psi(t+1) = \frac{(1-\psi(t+1)}{k} \to 0$ as $k \to \infty$.

4.5 Implications of the Result

4.5.1 Finding the optimal distribution over permutations of U

Since Perturbed-Greedy also chooses ranking orders over U through randomization, we can interpret it as a non-uniform Ranking, where it chooses permutations of U from the 'optimal' distribution. But we could have posed the following question, without the

knowledge of our algorithm: How do we find an optimal non-uniform distribution over permutations of U? As a start, let us consider the case of 2×2 graphs. By exhaustive search over all 2×2 graphs, we can figure out the best RANKING like algorithm for 2×2 graphs (Figure 9 shows the only two potentially 'hard' instances in 2×2 graphs). This algorithm picks the permutation (u_1, u_2) with probability $\frac{\alpha}{1+\alpha}$ and the permutation (u_2, u_1) with probability $\frac{1}{1+\alpha}$ (where $\alpha = b_{u_1}/b_{u_2}$), and then proceeds to match to the highest neighbor. This algorithm gives a factor of $\frac{\alpha^2+\alpha+1}{(\alpha+1)^2}$, which is minimized at $\alpha = 1$, giving a factor of 3/4 (in which case the algorithm is simply the same as RANKING).

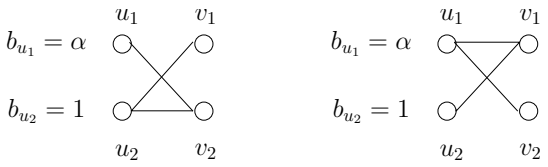


Figure 9: Canonical examples for 2×2 graphs.

An attempt to generalize this idea to larger graphs fails due to a blow-up in complexity. In general, we need a probability variable p_σ for every permutation σ of U. The expected weight of the matching produced by the algorithm on a graph G, is a linear expression $\text{ALG}_G(p_{\sigma_1}, p_{\sigma_2}, ...)$. Thus, the optimal distribution over permutations is given by the optimal solution of a linear program in the p_σ variables. But this LP has exponentially many variables (one per permutation) and constraints (one per "canonical graph instance"). Therefore, our algorithm can be thought of as solving this extremely large LP through a very simple process.

4.5.2 General capacities / Matching $u \in U$ multiple times

Consider the following generalization of the online vertex-weighted bipartite matching problem: Apart from a weight b_u, each vertex $u \in U$ has a capacity c_u such that u can be matched to *at most* c_u vertices in V. The capacities allow us to better model 'budgets' in many practical situations, *e.g.*, in online advertising. Our algorithm easily handles general capacities: For each $u \in U$, make c_u copies of u and solve the resulting instance with unit capacities: It is easy to verify that the solution is $\left(1 - \frac{1}{e}\right)$-approximate in expectation for the original problem with capacities.

4.5.3 Online budgeted allocation :- The *single bids* case vs. the *small bids* case

As noted earlier, the *single bids* case of the online budgeted allocation problem reduces to online vertex-weighted bipartite matching. Let us first define these problems.

Online budgeted allocation: We have n agents and m items. Each agent i specifies a monetary budget B_i and a bid b_{ij} for each item j. Items arrive online, and must be immediately allocated to an agent. If a set S of items is allocated to agent i, then the agent pays the minimum of B_i and $\sum_{j \in S} b_{ij}$. The objective is to maximize the total revenue of the algorithm.

Single bids case: Any bid made by agent i can take only two values: b_i or 0. In other words, all the non-zero bids of an agent are equal.

Corollary 28. *Online budgeted allocation with single bids reduces to online vertex-weighted bipartite matching, and hence* Perturbed-Greedy *achieves a competitive ratio of* $1 - 1/e$ *for this problem.*

Proof. Given an instance of online budgeted allocation where agent i has budget B_i and single bid value b_i, we will construct an input instance $G(U, V, E, \{b_u\}_{u \in U})$ of online vertex-weighted bipartite matching. The set V consists of one vertex corresponding to every item. The set U will contain one or more vertices for every agent.

For every agent i, let n_i be the largest integer such that $n_i b_i \leq B_i$ and let $r_i = B_i - n_i b_i$. Clearly, $r_i < b_i$. We will construct a set U_i of n_i vertices, each with weight b_i. In addition, if $r_i > 0$, then we will construct a vertex $\bar{u}_i$ with weight r_i and add it to U_i. For all $u \in U_i$ and $v \in V$, the edge $uv \in E$ if and only if agent i makes a non-zero bid on the item corresponding to v.

(1) Given a solution to the budgeted allocation problem where a set S_i of items is allocated to agent i, let us see how to construct a solution to the vertex-weighted matching problem with the same total value.

- If agent i pays a total of $|S_i| \cdot b_i$, then we know that $|S_i| \leq n_i$. Hence, for every item in S_i, we will match the corresponding vertex in V to a vertex in $U_i - \{\bar{u}_i\}$. Let R_i be the set of vertices in U_i thus matched. We have:

$$\sum_{u \in R_i} b_u \;=\; |R_i| \cdot b_i \;=\; |S_i| \cdot b_i$$

- If agent i pays a total amount strictly less than $|S_i| \cdot b_i$, then we know that: (a) $|S_i| \geq n_i + 1$, (b) $r_i > 0$ and (3) agent i pays the budget B_i. We can now choose any $n_i + 1$ items in S_i and match the corresponding vertices in V to the $n_i + 1$ vertices in U_i. The sum of the weights of matched vertices in U_i, $\sum_{u \in U_i} b_u = B_i$.

Summing over all i, the weight of the matching formed is equal to the total revenue of the budgeted allocation. Let OPT_A and OPT_M denote the values of the optimal solutions of the budgeted allocation and the vertex-weighted matching problems respectively. Then we conclude from the above discussion that:

$$\text{OPT}_M \;\geq \text{OPT}_A \tag{34}$$

(2) Given a solution to the vertex-weighted matching problem where a set $R \subseteq U$ of vertices is matched, let us see how to construct a solution to the budgeted allocation problem with at least the same total value. Let $R_i = R \cap U_i$. For every $v \in V$ that is matched to a vertex in R_i, we will allocate the corresponding item to agent i. Let S_i be the set of items allocated to agent i.

- If $|R_i| = |S_i| \leq n_i$, then agent i pays a total of $|S_i| \cdot b_i$ and we have:

$$\sum_{u \in R_i} b_u \;\leq\; |R_i| \cdot b_i \;=\; |S_i| \cdot b_i$$

- If on the other hand, $|R_i| = |S_i| = n_i + 1$ then agent i pays a total of B_i and we have:

$$\sum_{u \in R_i} b_u \;=\; \sum_{u \in U_i} b_u \;=\; B_i$$

Summing over all i, the total revenue of the budgeted allocation is at least the weight of the matching. Let ALG_M be the expected weight of the vertex-weighted matching constructed by Perturbed-Greedy and ALG_A be the expected value of the budgeted allocation constructed using the above scheme. From the above discussion, we conclude: Therefore,

$$\begin{aligned} \text{ALG}_A &\geq \text{ALG}_M \\ &\geq \left(1 - \frac{1}{e}\right) \text{OPT}_M \qquad (35) \\ &\geq \left(1 - \frac{1}{e}\right) \text{OPT}_A \end{aligned}$$

Here, equation (35) follows from the main result - Theorem 20 - and the last step uses equation (34). This completes our proof. □

Connection to the *small bids* case: Note that the *small bids* case ($b_{ij} \ll B_i$) studied in [47, 11] does not reduce to or from the *single bids* case. Yet, as it turns out, Perturbed-Greedy is equivalent to the algorithm of [47] - let us call it MSVV - on instances that belong to the intersection of the two cases. When every agent has a *single small bid* value, the problem corresponds to vertex-weighted matching with large capacities c_u for every vertex u. Recall that we handle capacities on $u \in U$ by making c_u copies $u_1, u_2, ..., u_{c_u}$ of u. For each of these copies, we choose a random $x_{u_i} \in [0, 1]$ uniformly and independently. In expectation, the x_{u_i}'s are uniformly distributed in the interval $[0, 1]$. Also observe that Perturbed-Greedy will match $u_1, u_2, ..., u_{c_u}$ in the increasing order of x_{u_i}'s, if at all. Therefore, at any point in the algorithm, if u_i is the unmatched copy of u with smallest x_{u_i} (and consequently highest multiplier $\psi(x_{u_i})$) then x_{u_i} is in expectation equal to the fraction of the capacity c_u used up at that point. But MSVV uses exactly the scaling factor $\psi(T)$ where T is the fraction of spent budget at any point. We conclude that in expectation, Perturbed-Greedy tends to MSVV as the capacities grow large, in the single small bids case.

It is important to see that this phenomenon is not merely a consequence of the common choice of function ψ. In fact, the function ψ is not a matter of choice at all - it is a by-product of both analyses (Refer to the remark at the end of Section 4.3). The fact that it happens

to be the exact same function seems to suggest that ψ is the 'right' function. Moreover, the analyses of the two algorithms do not imply one-another. Our variables are about expected gains and losses over a probability space, while the algorithm in [47] is purely deterministic.

This smooth 'interface' between the seemingly unrelated *single bids* and *small bids* cases hints towards the existence of a unified solution to the general online budgeted allocation problem.

CHAPTER V

OPEN AVENUES

Unifying computational and information theoretic hardness: A very interesting avenue of future work in light of our comments in Section 2.5.2 is the possibility of a hardness proof that makes *both* computational (*i.e.*, based on P $\neq$ NP) and information theoretic arguments. To our knowledge, no such proof is yet known for any problem. The need for such a technique is shown by our results for the multi-agent submodular shortest path problem. Our algorithm for this problem makes only polynomially many oracle calls, but needs to solve an NP-hard problem in order to match the lower bound! Ostensibly, a stricter hardness may apply if we also restrict the computational power of our algorithm.

Separation between truthful and non-truthful mechanisms in single parameter domains: As we mentioned in Chapter 3, for single parameter domains, it is known that *monotone* allocation rules characterize the set of all truthful mechanisms. An allocation rule or algorithm is said to be monotone if the allocation parameter of an agent ($f(S_i)$ in our case) is non-decreasing in his reported bid v_i. Unfortunately, often it is the case that good approximation algorithms known for a given class of valuation functions are not monotonic. It is an important and well-known open question in algorithmic mechanism design to resolve whether the design of monotone algorithms is fundamentally harder than the non-monotone ones. In other words, it is not known if, for single parameter problems, we can always convert any α-approximation algorithm into a truthful mechanism with the same factor. We believe that our problem is a suitable candidate to attack this question as it gives a lot of flexibility in defining the complexity of function f.

Towards resolution of the online budgeted allocation problem: In Chapter 4, we discussed connections between our vertex-weighted matching problem and online budgeted

allocation. Two pieces of this puzzle are known: The solution to the small bids case by [47] and our solution to the single bids case. The fact that these pieces fit together, as explained in Section 4.5.3, is encouraging as it strongly suggests the possibility of a unified solution.

As the authors of [47] explain, the role of the function ψ in their algorithm is to trade-off the effect of a higher bid with that of a larger remaining budget. On the other hand, we use the function ψ to choose the appropriate perturbation of bids. It is necessary to reconcile these two ideas in the unrestricted online budgeted allocation setting.

APPENDIX A

COMBINATORIAL AUCTIONS WITH PARTIALLY PUBLIC VALUATIONS

A.1 Greedy Allocation is Not Truthful

The greedy algorithm in our model works as follows: In each step it assigns one unallocated item j to a buyer i, where the pair (i, j) is chosen so as maximize the marginal gain in the objective function. That is, if buyer i had been allocated the set S of items before the current step, then $v_i\left(f(S \cup \{j\}) - f(S)\right)$ is maximized.

We will construct an example that adheres to our formulation of the TV Ad auctions problem. Consider an instance with two advertisers i_1 and i_2 and three ad-slots j_1, j_2, j_3. Suppose there are 10 viewers $k_1, ..., k_{10}$. Viewers k_1 to k_5 watch slot j_1, k_6 to k_{10} watch slot j_2 and k_3 to k_8 watch slot j_3. The public function f in this case is the coverage function: for a set S of slots $f(S)$ is the number of unique viewers who watch any slot in S. To prove that the greedy algorithm does not make monotonic allocations in this example, consider two cases:

1. $v_{i_1} = 1$ **and** $v_{i_2} = 1 + \epsilon$: In the first step, the greedy algorithm assigns the largest slot j_3 (with six viewers) to i_2. In the next two steps, it assigns both j_1 and j_2 to i_1. Therefore, the i_1 receives the set $\{j_3\}$ of total allocation value (not counting the private multiplier) 6.

2. $v_{i_1} = 1$ **and** $v_{i_2} = 1 - \epsilon$: In the first step, the greedy algorithm assigns the largest slot j_3 (with six viewers) to i_1. In the next two steps, it assigns both j_1 and j_2 to i_2. Therefore, the i_1 receives the set $\{j_1, j_2\}$ of total allocation value (not counting the private multiplier) 10.

Clearly, i_2 receives a larger allocation at a lower private valuation. Therefore, the greedy algorithm is not monotone.

APPENDIX B

ONLINE VERTEX-WEIGHTED BIPARTITE MATCHING AND SINGLE-BID BUDGETED ALLOCATIONS

B.1 Performance of GREEDY *and* RANKING

With non-equal weights, it is clearly preferable to match vertices with larger weight. This leads to the following natural algorithm.

Algorithm 5: GREEDY

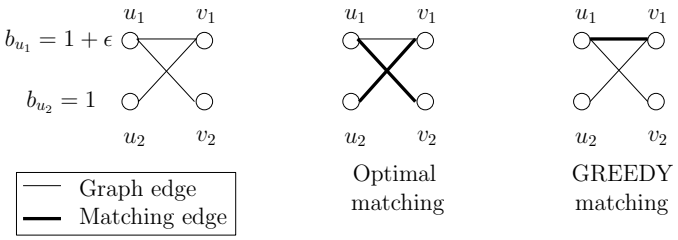
foreach *arriving* $v \in V$ **do**
Match v to the unmatched neighbor in u which maximizes b_u (breaking ties arbitrarily);

It is not hard to show that GREEDY achieves a competitive ratio of at least $\frac{1}{2}$.

Lemma 29. GREEDY *achieves a competitive ratio of 1/2 in vertex-weighted online bipartite matching.*

Proof. Consider an optimal offline matching, and a vertex $u \in U$ that is matched in the optimal offline matching but not in the greedy algorithm. Now look at a vertex $u^* \in V$ that is matched to the vertex u in the optimal matching. In GREEDY, u^* must have been matched to a vertex $u^{'} \in U$, s.t. $b_u \leq b_{u'}$, since u was unmatched when u^* was being matched. So we'll charge the loss of b_u to $u^{'}$. Note that each $u^{'}$ does not get charged more than once – it is charged only by the optimal partner of its partner in the algorithm's matching. Thus the loss of the algorithm is no more than the value of the matching output by the algorithm. Hence the claim. □

In fact, this factor 1/2 is tight for GREEDY as shown by an instance consisting of many copies of the following gadget on four vertices, with $u_1, u_2 \in U$ and $v_1, v_2 \in V$. As $\epsilon \to 0$, the competitive ratio of GREEDY tends to $\frac{1}{2}$.



Notice that this counter-example relies on weights being roughly equal. We, however, know that RANKING has an expected competitive ratio of $(1 - 1/e)$ when the weights are equal. On the other hand, if the weights are very different, i.e. ϵ is large, in the above example, then GREEDY provides a good competitive ratio. At the same time, if we exchanged the weights on the two vertices in the example to be $b_{u_1} = 1$ and $b_{u_2} = 1 + \epsilon$, then as ϵ grows large, the expected competitive ratio of RANKING drops to $\frac{1}{2}$ and on larger examples, it can be as low as $\frac{1}{n}$. To summarize, GREEDY tends to perform well when the weights are highly skewed and RANKING performs well when the weights are roughly equal.

B.2 A Lower Bound for Randomized Algorithms with Edge Weights

In this section, we will sketch the proof of a lower bound for the competitive ratio of a randomized algorithm, when the graph $G(U, V, E)$ has edge weights and our objective is to find a matching in G with maximum total weight of edges. Previous studies of this problem have only mentioned that no constant factor can be achieved when the vertices in V arrive in an online manner. However, we have not been able to find a proof of this lower bound for randomized algorithms in any literature. We prove the result when the algorithm is restricted to be scale-free. A scale-free algorithm in this context produces the exact same matching when all the edge weights are multiplied by the same factor.

Consider a graph $G(U, V, E)$ such that U contains just one vertex u and each vertex in $v \in V$ has an edge to u of weight b_v. Fix $v_1, v_2, ...$ to be the order in which the vertices of V arrive online. By Yao's principle, it suffices for us to produce a probability distribution over $b_{v_1}, b_{v_2}, ...$ such that no deterministic algorithm can perform well in expectation. We will denote the vector of edge weights in the same order in which the corresponding vertices in V arrive, *i.e.* $(b_{v_1}, b_{v_2}, ...)$ and so on. Consider the following n vectors of edge weights: For every $1 \leq i \leq n$, $\mathbf{b}_i = (D^i, D^{i+1}, ..., D^n, 0, 0, ...)$ and so on, where D is a sufficiently large

number. Suppose our input distribution chooses each one of these n vectors of edge weights with equal probability.

Clearly, regardless of the vector which is chosen, $\mathrm{OPT}(G) = D^n$. Since an algorithm is assumed to be scale-free and online, it makes the exact same decisions after the arrival of first k vertices for each of the edge weight vectors $\mathbf{b}_j$, $1 \leq j \leq k$. Therefore, it cannot distinguish between $\mathbf{b}_1, ..., \mathbf{b}_k$ after just k steps. Hence, we can characterize any algorithm by the unique k such that it matches the k'th vertex in V with a positive weight edge.

Let ALG be any deterministic algorithm that matches the k'th incoming vertex with a positive weight edge to u. Then the expected weight of the edge chosen by ALG is $\frac{1}{n}\sum_{i>k} D^i$. Since D is large, this is at most $\frac{c}{n}\mathrm{OPT}(G)$, where c is some constant. Applying Yao's principle, we conclude that the competitive ratio of the best scale-free randomized algorithm for online bipartite matching with edge weights is $O\left(\frac{1}{n}\right)$.

REFERENCES

[1] Aggarwal, G., Goel, G., Karande, C., and Mehta, A., "Online vertex-weighted bipartite matching and single-bid budgeted allocations," *CoRR*, 2010.

[2] Archer, A., Papadimitriou, C., Talwar, K., and Tardos, E., "An approximate truthful mechanism for combinatorial auctions with single parameter agents," in *SODA '03: Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms*, pp. 205–214, 2003.

[3] Babaioff, M., Immorlica, N., and Kleinberg, R., "Matroids, secretary problems, and online mechanisms," in *SODA '07: Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pp. 434–443, 2007.

[4] Babaioff, M., Lavi, R., and Pavlov, E., "Single-value combinatorial auctions and algorithmic implementation in undominated strategies," *J. ACM*, vol. 56, no. 1, pp. 1–32, 2009.

[5] Balcan, M.-F., Blum, A., Hartline, J. D., and Mansour, Y., "Reducing mechanism design to algorithm design via machine learning," *J. Comput. Syst. Sci.*, vol. 74, no. 8, pp. 1245–1270, 2008.

[6] Balcan, M.-F. and Harvey, N., "Learning submodular functions," *CoRR*, 2010.

[7] Bar-Yehuda, R. and Even, S., "A linear time approximation algorithm for the weighted vertex cover problem," *Journal of Algorithms*, vol. 2, pp. 198–203, 1981.

[8] Birnbaum, B. and Mathieu, C., "On-line bipartite matching made simple," *SIGACT News*, vol. 39, no. 1, pp. 80–87, 2008.

[9] Blumrosen, L. and Nisan, N., *Algorithmic Game Theory*, ch. 11. Cambridge University Press, 2007.

[10] Bollobas, B., *Random Graphs*. Cambridge University Press, 2001.

[11] Buchbinder, N., Jain, K., and Naor, J. S., "Online primal-dual algorithms for maximizing ad-auctions revenue," in *ESA'07: Proceedings of the 15th annual European conference on Algorithms*, pp. 253–264, 2007.

[12] Buchfuhrer, D., Dughmi, S., Fu, H., Kleinberg, R., Mossel, E., Papadimitriou, C., Schapira, M., Singer, Y., and Umans, C., "Inapproximability for vcg-based combinatorial auctions," in *SODA '10: Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete algorithms*, pp. 518–536, 2010.

[13] Calinescu, G., Chekuri, C., Pál, M., and Vondrák, J., "Maximizing a submodular set function subject to a matroid constraint (extended abstract)," in *IPCO '07: Proceedings of the 12th international conference on Integer Programming and Combinatorial Optimization*, pp. 182–196, 2007.

[14] Devenur, N. R. and Hayes, T. P., "The adwords problem: online keyword matching with budgeted bidders under random permutations," in *EC '09: Proceedings of the tenth ACM conference on Electronic commerce*, pp. 71–78, 2009.

[15] Dimitrov, N. B. and Plaxton, C. G., "Competitive weighted matching in transversal matroids," in *ICALP '08: Proceedings of the 35th international colloquium on Automata, Languages and Programming, Part I*, pp. 397–408, 2008.

[16] Dinur, I. and Safra, S., "On the hardness of approximating minimum vertex cover," *Annals of Mathematics*, vol. 162, no. 1, pp. 439–486, 2005.

[17] Dobzinski, S., "Two randomized mechanisms for combinatorial auctions," in *APPROX '07: Proceedings of the 10th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems*, pp. 89–103, 2007.

[18] Dobzinski, S. and Nisan, N., "Limitations of vcg-based mechanisms," in *STOC '07: Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*, pp. 338–344, 2007.

[19] Dobzinski, S., Nisan, N., and Schapira, M., "Approximation algorithms for combinatorial auctions with complement-free bidders," in *STOC '05: Proceedings of the thirty-seventh annual ACM symposium on Theory of computing*, pp. 610–618, 2005.

[20] Dobzinski, S. and Schapira, M., "An improved approximation algorithm for combinatorial auctions with submodular bidders," in *SODA '06: Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, pp. 1064–1073, 2006.

[21] Feige, U., Mirrokni, V. S., and Vondrak, J., "Maximizing non-monotone submodular functions," in *FOCS '07: Proceedings of the 48th Annual IEEE Symposium on Foundations of Computer Science*, pp. 461–471, 2007.

[22] Feige, U. and Vondrak, J., "Approximation algorithms for allocation problems: Improving the factor of 1 - 1/e," in *FOCS '06: Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science*, pp. 667–676, 2006.

[23] Feldman, J., Mehta, A., Mirrokni, V., and Muthukrishnan, S., "Online stochastic matching: Beating 1-1/e," in *FOCS '09: Proceedings of the 2009 50th Annual IEEE Symposium on Foundations of Computer Science*, pp. 117–126, 2009.

[24] Garg, R., Kumar, V., and Pandit, V., "Approximation algorithms for budget-constrained auctions," in *APPROX '01/RANDOM '01: Proceedings of the 4th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems and 5th International Workshop on Randomization and Approximation Techniques in Computer Science*, pp. 102–113, 2001.

[25] Goel, G., Karande, C., Tripathi, P., and Wang, L., "Approximability of combinatorial problems with multi-agent submodular cost functions," in *FOCS '09: Proceedings of the 2009 50th Annual IEEE Symposium on Foundations of Computer Science*, pp. 755–764, 2009.

[26] Goel, G., Karande, C., and Wang, L., "Single parameter combinatorial auctions with partially public valuations," in *SAGT '10: Proceedings of the Third Annual International Symposium on Algorithmic Game Theory*, 2010.

[27] Goel, G. and Mehta, A., "Online budgeted matching in random input models with applications to adwords," in *SODA '08: Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms*, pp. 982–991, 2008.

[28] Goemans, M. X., Harvey, N. J. A., Iwata, S., and Mirrokni, V., "Approximating submodular functions everywhere," in *SODA '09: Proceedings of the twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 535–544, 2009.

[29] Hartline, J. D. and Lucier, B., "Bayesian algorithmic mechanism design," in *STOC '10: Proceedings of the 42nd ACM symposium on Theory of computing*, pp. 301–310, 2010.

[30] Hayrapetyan, A., Swamy, C., and Tardos, E., "Network design for information networks," in *SODA '05: Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*, pp. 933–942, 2005.

[31] Holzman, R., Kfir-dahav, N., Monderer, D., and Tennenholtz, M., "Bundling equilibrium in combinatorial auctions," *Games and Economic Behavior*, vol. 47, pp. 104–123, April 2001.

[32] Iwata, S., "A faster scaling algorithm for minimizing submodular functions," *SIAM Journal of Computing*, vol. 32, no. 4, pp. 833–840, 2003.

[33] Iwata, S., "Submodular function minimization," *Mathematical Programming*, vol. 112, no. 1, pp. 45–64, 2008.

[34] Iwata, S., Fleischer, L., and Fujishige, S., "A combinatorial strongly polynomial algorithm for minimizing submodular functions," *J. ACM*, vol. 48, no. 4, pp. 761–777, 2001.

[35] Iwata, S. and Nagano, K., "Submodular function minimization under covering constraints," in *FOCS '09: Proceedings of the 2009 50th Annual IEEE Symposium on Foundations of Computer Science*, pp. 671–680, 2009.

[36] Iwata, S. and Orlin, J. B., "A simple combinatorial algorithm for submodular function minimization," in *SODA '09: Proceedings of the Nineteenth Annual ACM -SIAM Symposium on Discrete Algorithms*, pp. 1230–1237, 2009.

[37] Kalyanasundaram, B. and Pruhs, K., "Online weighted matching," *J. Algorithms*, vol. 14, no. 3, pp. 478–488, 1993.

[38] Karp, R., Vazirani, U., and Vazirani, V., "An optimal algorithm for online bipartite matching," in *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing*, pp. 352–358, 1990.

[39] Khot, S., "On the power of unique 2-prover 1-round games," in *STOC '02: Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pp. 767–775, 2002.

[40] Khot, S., Lipton, R. J., Markakis, E., and Mehta, A., "Inapproximability results for combinatorial auctions with submodular utility functions," *Algorithmica*, vol. 52, no. 1, pp. 3–18, 2008.

[41] Khot, S. and Regev, O., "Vertex cover might be hard to approximate to within 2-epsilon," *J. Comput. Syst. Sci.*, vol. 74, no. 3, pp. 335–349, 2008.

[42] Korula, N. and Pál, M., "Algorithms for secretary problems on graphs and hypergraphs," in *ICALP '09: Proceedings of the 36th International Colloquium on Automata, Languages and Programming*, pp. 508–520, 2009.

[43] Krohn, E. and Varadarajan, K., "Private communication." 2007.

[44] Lavi, R. and Swamy, C., "Truthful and near-optimal mechanism design via linear programming," in *FOCS '05: Proceedings of the 46th Annual IEEE Symposium on Foundations of Computer Science*, pp. 595–604, 2005.

[45] Lehmann, B., Lehmann, D., and Nisan, N., "Combinatorial auctions with decreasing marginal utilities," in *Proceedings of the 3rd ACM conference on Electronic Commerce*, pp. 18 –28, 2001.

[46] Lehmann, D., Oćallaghan, L. I., and Shoham, Y., "Truth revelation in approximately efficient combinatorial auctions," *J. ACM*, vol. 49, no. 5, pp. 577–602, 2002.

[47] Mehta, A., Saberi, A., Vazirani, U., and Vazirani, V., "Adwords and generalized online matching," *J. ACM*, vol. 54, no. 5, p. 22, 2007.

[48] Mirrokni, V. S., Schapira, M., and Vondrák, J., "Tight information-theoretic lower bounds for welfare maximization in combinatorial auctions," in *ACM Conference on Electronic Commerce*, pp. 70–77, 2008.

[49] Mu'alem, A. and Nisan, N., "Truthful approximation mechanisms for restricted combinatorial auctions: extended abstract," *Games and Economic Behavior*, vol. 64, no. 2, pp. 612–631, 2008.

[50] Nisan, N., Bayer, J., Chandra, D., Franji, T., Gardner, R., Matias, Y., Rhodes, N., Seltzer, M., Tom, D., Varian, H., and Zigmond, D., "Google's auction for tv ads," in *ICALP '09: Proceedings of the 36th Internatilonal Collogquium on Automata, Languages and Programming*, pp. 309–327, 2009.

[51] Orlin, J. B., "A faster strongly polynomial time algorithm for submodular function minimization," in *IPCO '07: Proceedings of the 12th international conference on Integer Programming and Combinatorial Optimization*, pp. 240–251, 2007.

[52] Papadimitriou, C., Schapira, M., and Singer, Y., "On the hardness of being truthful," in *FOCS '08: Proceedings of the 2008 49th Annual IEEE Symposium on Foundations of Computer Science*, pp. 250–259, 2008.

[53] Schrijver, A., "A combinatorial algorithm minimizing submodular functions in strongly polynomial time," *J. Comb. Theory, Ser. B*, vol. 80, no. 2, pp. 346–355, 2000.

[54] Seshadhri, C. and Vondrak, J., "Is submodularity testable?," *CoRR*, 2010.

[55] Sharma, Y., Swamy, C., and Williamson, D. P., "Approximation algorithms for prize collecting forest problems with submodular penalty functions," in *SODA '07: Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pp. 1275–1284, 2007.

[56] Sviridenko, M., "A note on maximizing a submodular set function subject to a knapsack constraint," *Oper. Res. Lett.*, vol. 32, no. 1, pp. 41–43, 2004.

[57] Svitkina, Z. and Fleischer, L., "Submodular approximation: Sampling-based algorithms and lower bounds," in *FOCS '08: Proceedings of the 2008 49th Annual IEEE Symposium on Foundations of Computer Science*, pp. 697–706, 2008.

[58] Svitkina, Z. and Tardos, E., "Facility location with hierarchical facility costs," in *SODA '06: Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, pp. 153–161, 2006.

[59] Vondrak, J., "Optimal approximation for the submodular welfare problem in the value oracle model," in *STOC '08: Proceedings of the 40th annual ACM symposium on Theory of computing*, pp. 67–74, 2008.

[60] Wolsey, L. A., "An analysis of the greedy algorithm for the submodular set covering problem," *Combinatorica*, vol. 2, no. 4, pp. 385–393, 1982.