ALGORITHMIC ASPECTS OF CONNECTIVITY, ALLOCATION AND DESIGN PROBLEMS

A Thesis
Presented to
The Academic Faculty

by

Deeparnab Chakrabarty

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

College of Computing
Georgia Institute of Technology
August 2008

ALGORITHMIC ASPECTS OF CONNECTIVITY, ALLOCATION AND DESIGN PROBLEMS

Approved by:

Professor Vijay V. Vazirani, Advisor
College of Computing
Georgia Tech

Professor William Cook
School of Industrial Systems and Engineering
Georgia Tech

Professor Robin Thomas
School of Mathematics
Georgia Tech

Professor Adam Kalai
College of Computing
Georgia Tech

Professor Prasad Tetali
School of Mathematics
Georgia Tech

Date Approved: 15th May 2008

To those who probably will understand the least, but matter the most.

ACKNOWLEDGEMENTS

First and foremost, I thank my advisor, Vijay Vazirani, for his constant support, advice and help throughout these five years. I hope I have imbibed an ϵ of the infinite energy with which he does research. Thanks Vijay for everything!

I thank Prasad Tetali for patiently listening to me ranting about my work and always advising me on how to proceed further. I will surely miss such an ideal sounding board.

I thank Robin Thomas, William Cook and Adam Kalai for being on my committee. Thanks Robin for a careful reading of a paper of mine, Prof. Cook for being a careful reader of my thesis, and Adam for the chats and the margaritas!

Many thanks to all the professors in Georgia Tech for their excellent lectures, both in and out of classes. The amount I have learnt in these five years is immeasurable.

I am indebted to ACO, the excellent program I was in, and the College of Computing, for the excellent facilities and support it provided me throughout my stay here. Special thanks to Dani Denton for always taking care of the paperwork, which I am horrible at.

Thanks to all the excellent graduate students in theory at Georgia Tech who are more friends than colleagues. Some of them are my co-authors — Nisheeth Vishnoi, Aranyak Mehta, Nikhil Devanur, Gagan Goel and Atish Das Sarma. Without them none of this would have been possible. A very special thanks to Nikhil Devanur for the endless hours we spent discussing work, life and everything under the sun.

I thank HP-Labs, Palo Alto, and my mentors Dr. Yunhong Zhou and Dr. Rajan Lukose for a wonderful summer in California.

Many thanks to Prof. Abhiram Ranade of IIT-Bombay, my alma-mater, for starting me off on my research career. A special thanks to Viswanath Nagarajan for being a friend and colleague since my undergraduate days.

My life as a graduate student would have been a million-folds more stressful, were it not for the only Indian fraternity in Atlanta — *Rama Papa Kappa*. Thanks to all the members for always putting things into perspective.

Words cannot describe my gratitude towards the people dearest to me — Harini, Tua, my family, my mother and my father. Their belief in me, although at times irrational, has been vital for this work. I dedicate this thesis to them.

TABLE OF CONTENTS

LIST OF FIGURES

SUMMARY

Most combinatorial optimization problems are NP-hard, which implies that under well-believed complexity assumptions, there exist no polynomial time algorithms to solve them. To cope with the NP-hardness, *approximation algorithms* which return solutions close to the optimal, have become a rich field of study. One successful method for designing approximation algorithms has been to model the optimization problem as an integer program and then using its polynomial time solvable linear programming relaxation for the design and analysis of such algorithms. Such a technique is called the *LP-based* technique.

In this thesis, we study the algorithmic aspects of three classes of combinatorial optimization problems using LP-based techniques as our main tool.

Connectivity Problems: We study the *Steiner tree* problem and devise new linear programming relaxations for the problem. We show an equivalence of our relaxation with the well studied *bidirected cut* relaxation for the Steiner tree problem. Furthermore, for a class of graphs called quasi-bipartite graphs, we improve the best known upper bound on the integrality gap from 3/2 to 4/3. Algorithmically, we obtain fast and simple approximation algorithms for the Steiner tree problem on quasi-bipartite graphs.

Allocation Problems: We study the *budgeted allocation* problem of allocating a set of indivisible items to a set of agents who bid on it but possess a hard budget constraint more than which they are unwilling to pay. This problem is a special case of submodular welfare maximization. We use a natural LP relaxation for the problem and improve the best known approximation factor for the problem from $\simeq 0.632$ to 3/4. We also improve the inapproximability factor of the problem to 15/16 and use our techniques to show inapproximability results for many other allocation problems.

We also study *online* allocation problems where the set of items are unknown and appear one at a time. Under some necessary assumptions we provide online algorithms for many problems which attain the (almost) optimal competitive ratio. Both these works have applications in the area of budgeted auctions, the most famous of which are the sponsored search auctions hosted by search engines on the Internet.

Design Problems: We formally define and study *design problems* which ask how the weights of an input instance can be designed, so that the minimum (or maximum) of a certain function of the input can be maximized (respectively, minimized). We show if the function can be approximated to any factor α, then the optimum design can be approximated to the same factor.

We also show that (max-min) design problems are dual to packing problems. We use the framework developed by our study of design problems to obtain results about fractionally packing Steiner trees in a "black-box" fashion. Finally, we study integral packing of spanning trees and provide an alternate proof of a theorem of Nash-Williams and Tutte about packing spanning trees.

CHAPTER I

INTRODUCTION

The field of optimization deals with finding an element from a set which optimizes, for instance minimizes or maximizes, a certain function over all elements in the set. Discrete optimization, also called combinatorial optimization, is the sub-field of optimization when the set of elements is finite. Problems in combinatorial optimization are interesting if the set is *concisely represented* and thus going over all the elements of the set, whose number can be exponentially larger than the *size* of the representation, is no longer an interesting option, and more efficient methods need to be found. For instance, a typical combinatorial optimization problem is: Given a finite graph G with vertex set V and edge set E, find a path between two specified vertices s and t which uses the minimum number of edges. Here, the set of solutions is all the paths between s and t, however all these paths are represented by the graph G which requires defining its two sets, the total size of which is $(|V| + |E|)$. On the other hand the total number of paths between s and t could be almost as large as $2^{|V|}$. A combinatorial optimization problem is said to be in the class P(Pfor polynomial), if it can be solved in time which is bounded by a polynomial in the size of the representation. For instance, the shortest-path problem mentioned in the above paragraph has a polynomial time algorithm (given by Edsger Dijkstra) and is thus in P. Apart from the obvious advantage of getting faster algorithms, research in combinatorial optimization problems in Phave led to a deeper understanding of the problems themselves. An excellent and comprehensive reference is the three-volume monograph of Alexander Schrijver [88].

However, many combinatorial optimization problems are NP-hard, that is, if one believe the P$\neq$NPconjecture, then such problems do not have polynomial time algorithms. For instance, going back to the above example, if instead of asking for the shortest path, we wanted to find the longest path between s and t which repeats no vertices, the problem becomes NP-hard.

How does one cope with NP-hardness? There are many ways researchers have done this including design of heuristics which work well in practice, average case analysis of algorithms which work well assuming some distribution on the input and *approximation algorithms* which find solutions which are not optimal, but are guaranteed to be not too bad either. In this thesis, we take the third way of coping with NP-hardness and investigate approximation algorithms for three broad classes of problems: connectivity problems, allocation problems and design/packing problems.

Throughout the thesis, we assume a certain acquaintance with terms, notations and concepts related to graphs, optimization and algorithms. We provide a few preliminaries

regarding the same in the appendix.

1.1 Approximation Algorithms

Given an optimization problem Π, we call an algorithm $\mathcal{A}$ a polynomial time approximation algorithm if it runs in polynomial time and returns a solution to the problem "close" to the optimal. If the problem is a minimization problem, then the algorithm $\mathcal{A}$ is called a $\rho()$-factor approximation algorithm, where $\rho : \mathbb{R} \to \mathbb{R}$ is a function taking reals to reals, if for any instance I of the optimization problem Π, the solution returned by the algorithm $\mathcal{A}$ is guaranteed to have cost

$$\mathcal{A}(I) \leq \rho(|I|) \cdot \mathrm{OPT}(I)$$

where $\mathrm{OPT}(I)$ is the value of the optimum solution to I and $|I|$ is the size of the representation of the instance I. Note that $\rho(|I|) \geq 1$ as one cannot possibly do better than the optimum. Furthermore $\mathcal{A}$ runs in time bounded by a polynomial in $|I|$. Similarly, if Π were a maximization problem, the definition would be the same with the inequality reversed, and $\rho()$ would be less than 1.

For instance, for the NP-hard longest path problem defined above consider the algorithm $\mathcal{A}$ which returns *any* path from s to t (we assume the graph is connected). Such an algorithm can be easily implemented in polynomial time. Since the longest path itself can be as long as n, the number of vertices, the algorithm $\mathcal{A}$ is a $\rho(n) = \frac{1}{n}$-factor algorithm.

Obviously, the approximation algorithm given above is in some sense a *trivial* approximation algorithm and opens up the question whether there are better approximation algorithms for the same. Such a case can be made for any NP-hard problem and this is what concerns the theory of approximation algorithms: determining the *approximibility*, that is the best approximation factor, of a given problem Π.

How does one find the factor of an approximation algorithm? Note that to show that $\mathcal{A}$ is a ρ-factor approximation algorithm, we need to show for every instance $\mathcal{A}(I) \leq \rho \cdot \mathrm{OPT}(I)$ (we abuse notation and omit the dependence of ρ on $|I|$). However, if Π is an NP-hard problem, evaluating $\mathrm{OPT}(I)$ itself is NP-hard. One way which is almost always used to prove that $\mathcal{A}$ is a ρ-factor algorithm is to find a bound on $\mathrm{OPT}(I)$ (a lower bound if Π is a minimization problem, an upper if Π is maximization) which can be found in polynomial time. For instance, for the longest path problem described above, the number of vertices forms an upper bound on the longest path and indeed the algorithm compares with this upper bound, rather than the true longest path itself.

A *mantra* for finding better approximation algorithms is to find better lower/upper bounds. One such method of obtaining better (efficiently computable) bounds is to model the combinatorial optimization problem as an *integer program* and then using the solution of its *linear programming relaxation* as the bound.

1.1.1 LP relaxations and Approximation Algorithms

For the purpose of exposition, let us pick one of the most fundamental NP-hard combinatorial optimization problems: the minimum vertex cover problem. Given a graph $G(V, E)$ and a weight vector $w : V \to \mathbb{R}_+$, the minimum weight vertex cover problem is to find a subset of vertices $W \subseteq V$ of minimum weight such that every edge in E has at least one end point in W.

Most combinatorial optimization problems can be cast as an integer program. For instance, for the vertex cover problem the following integer program gives the optimum solution to the problem.

$$\mathrm{OPT}(G) = \min\{\sum_{v \in V} w(v)x_v : \quad \forall (u,v) \in E,\ x_u + x_v \geq 1;\ \ \forall v \in V,\ x_v \in \{0,1\}\ \ \} \qquad (1)$$

The variable $x_v \in \{0, 1\}$ is an *indicator variable* for W: $x_v = 1$ iff $v \in W$. The constraint implies that each edge has at least one end point in W. Thus the integer program exactly corresponds to the minimum vertex cover problem.

However, since the vertex cover is NP-hard, finding the solution to program (1) is NP-hard as well. But if we replace the constraint $x_v \in \{0, 1\}$ by $1 \geq x_v \geq 0$, the resulting program is a linear program and one can solve it in polynomial time (either using the ellipsoid method [58, 46] or interior point methods [56]).

$$\mathrm{LP}(G) = \min\{\sum_{v \in V} w(v)x_v : \quad \forall (u,v) \in E,\ x_u + x_v \geq 1;\ \ \forall v \in V,\ 1 \geq x_v \geq 0\ \ \} \qquad (2)$$

On the flip side, the solutions do not correspond to vertex cover solutions. Nevertheless, since any integral solution (that is, those corresponding to the vertex cover solution) is a valid solution to the linear program, the optimum of the linear program is a *lower bound* on the minimum vertex cover. That is, for any graph G, $\mathrm{LP}(G) \leq \mathrm{OPT}(G)$. Thus, if one could develop a polynomial time algorithm which given a graph G could return a vertex cover of size $\mathcal{A}(G) \leq \rho \cdot \mathrm{LP}(G)$, $\mathcal{A}$ would a ρ-factor approximation algorithm.

1.1.2 Approximation via LP-rounding algorithms

Apart from providing a lower bound on the optimum solution, the LP relaxation can actually be used to obtain the approximation algorithms themselves. Such approximation algorithms are called *LP-based* approximation algorithms. One of the classes of LP-based approximation algorithms are *LP-rounding* algorithms which take the optimal LP-solution and *round* a set of the variables to obtain a valid solution. For instance, let $\{x^*_v\}_{v \in V}$ be an optimal solution to the LP relaxation LP (2) for the vertex cover solution. Consider the following algorithm $\mathcal{A}$: Pick all vertices v with $x^*_v \geq 1/2$ in W.

To see W is a valid vertex cover, consider any edge (u, v). From the constraint in the LP, $x_u + x_v \geq 1$, one of the two variables must be at least a $1/2$ and thus picked in W. Moreover, the weight of the vertices in W must be at most $2 \cdot \mathrm{LP}(G)$ since for each vertex picked v in W, the LP pays at least $w(v)/2$ while the algorithm pays $w(v)$. Thus the algorithm is a polynomial time 2-factor algorithm.

LP-rounding algorithms have been successful in giving the best known approximation algorithms for a host of problems. The LP-rounding technique itself has many different variants worth mentioning. The example given above is a *one-shot* rounding which solves the LP once and rounds a particular subset of variables to get the solution. There are examples which use the fact that an *extreme-point optimal solution* of an LP can be found and use the properties of the extreme-point solution to obtain algorithms. An example is the algorithm for the minimum makespan machine scheduling problem of Lenstra, Shmoys and Tardos [71] (also see Chapter 3). An interpretation of the LP optimum variables as *probabilities* of rounding the variable has proven to give approximation algorithms for multicommodity flows and routing [83]. Such a method is called *randomized rounding*. More recently, Jain [51] devised the *iterated rounding method* which rounds variables one at a time and re-solves the LP again on the residual problem after each rounding step. Jain used this procedure to give a 2 factor algorithm for the Steiner network problem and very recently, the method was used to obtain the best known approximation algorithms for degree bounded network design [90, 68, 8, 69].

1.1.3 Approximation via Primal-Dual algorithms

Another class of LP-based approximation algorithms are *primal-dual* algorithms which not only use the LP-relaxation, but also its *dual*. For instance, one can take the dual of the LP-relaxation LP (2), which by strong LP-duality gives:

$$\mathrm{LP}(G) = \max\{ \sum_{(u,v)\in E} y_{uv} : \quad \forall v \in V,\ y(\delta(v)) \leq w(v); \quad \forall (u,v) \in E,\ y_{uv} \geq 0\} \tag{3}$$

Call a vertex v *tight* if the constraint corresponding to it in LP (3) holds with equality: $y(\delta(v)) = w(v)$. Consider the following algorithm $\mathcal{A}$: Initialize the set W to be empty and let $y_{uv} = 0$ for all edges (u, v). Raise the dual variables y_{uv} for all edges which are not incident to any tight vertex till some new vertex goes tight. If the vertex v goes tight, put v in W and repeat till all edges are incident to tight vertices.

The set W is precisely the set of tight vertices and thus forms a vertex cover in the end since all edges are incident to tight vertices. Let $\{y_{uv}\}_{(u,v)\in E}$ be the final dual solution returned by the algorithm. From the running of the algorithm it is clear that y is a feasible solution for LP (3). Thus, $\mathrm{LP}(G) \geq \sum_{(u,v)\in E} y_{uv}$. We show that $\mathcal{A}$ is a 2-factor algorithm by showing that the weight of the solution picked is less than $2\sum_{(u,v)\in E} y_{uv}$ which is $\leq 2\mathrm{LP}(G) \leq 2\mathrm{OPT}(G)$. This follows from the fact that all vertices in W are tight:

$\sum_{v \in W} w(v) = \sum_{v \in W} y(\delta(v)) \leq \sum_{(u,v) \in E} 2y_{u,v}$ where the last inequality follows from the fact that each edge has at most 2 end-points in W.

One of the advantages of primal-dual algorithms is that one does not need to solve any LP and thus gives a purely combinatorial algorithm. However on the flip-side, these algorithms are hard to develop since they typically use only "local information" and designing such algorithms require considerable amount of finesse. We refer the reader to the book of Vazirani [93], and the survey by Goemans and Williamson [45] for expositions on the primal-dual method in approximation algorithms.

1.1.4 Limitations of an LP relaxation

Once again, this raises the following question: Can one obtain a better approximation algorithm for the minimum vertex cover problem *using* the lower bound suggested by LP (2)? In fact, one cannot as the following example shows. Consider the complete graph K_n on n vertices with unit weight on each vertex. Note that *any* vertex cover will use at least $n-1$ vertices. However, the solution $x_v = 1/2$ for all vertices $v \in V$ is still a feasible solution of weight $n/2$. Thus *any* algorithm using this LP-bound cannot give a factor better than $2 - 1/n$.

Thus the example of the complete graph on n vertices demonstrates the limitation of the LP-relaxation LP (2). This limitation is captured in the following definition of the *integrality gap* of an LP-relaxation for an optimization problem Π.

Definition 1 *Given a minimization problem Π and given an LP-relaxation LP_Π for the problem, the integrality gap of the LP-relaxation is defined as*

$$\mathcal{I}(LP_\Pi) := \sup_I \frac{OPT(I)}{LP_\Pi(I)}$$

where the supremum is taken over all instances I of the minimization problem.

For maximization problems, the integrality gap is defined similarly with the supremum replaced by an infimum.

It is clear that given any minimization problem Π and an LP-relaxation LP_Π for it, the best approximation one can prove using the LP-optimum as a lower bound is at least $\mathcal{I}(\mathrm{LP}_\Pi)$. For instance the integrality gap of the LP-relaxation of the vertex cover problem described above is 2.

1.1.5 Hardness of Approximation

As we saw, proving that the integrality gap of an LP relaxation for the vertex cover is 2 shows that no better than a 2-factor approximation is possible using that particular relaxation. However, this does not preclude the presence of *another* LP relaxation with a possibly smaller integrality gap which might be used to obtain better approximation algorithms.

A stronger evidence of inapproximability is provided if one shows that it is NP-hard to approximate a problem better than a factor ρ. Such a breakthrough was made in the early 1990's with the PCP theorem [6] which can be equivalently stated as:

Theorem 1.1.1 ([6]) *Given a 3SAT formula $\phi = C_1 \wedge C_2 \cdots \wedge C_m$, where each C_i is a disjunction of 3 literals, there exists a constant $\delta < 1$, such that it is NP-hard to distinguish between the two cases: (YES) There exists an assignment of variables such that all clauses are satisfied, and (NO): No assignment of variables satisfy more than δ fraction of the clauses.*

Note that the above theorem implies that no factor better than δ is possible for the MAX-3SAT problem which given a formula asks for an assignment satisfying the maximum number of clauses. Such a result is called an *hardness of approximation* result. The above theorem opened the floodgates for hardness of approximation results, notable among which is the work of Håstad [47] who showed many optimal inapproximability results, that is, hardness of approximation factors which matched the factors of known approximation algorithms.

Recently, Khot[59] conjectured a strengthening of the PCP theorem which is called the Unique Games Conjecture (UGC). Assuming this conjecture, it was proved [61] that it is NP-hard to approximate the minimum vertex cover problem to a factor better than $2 - \epsilon$, for any $\epsilon > 0$. Without assuming this conjecture, the best known hardness factor for the minimum vertex cover problem is 1.36 [30].

1.2 Contributions, Organization and Credits of this Thesis

In this thesis, we investigate algorithmic aspects of three classes of combinatorial optimization problems. The three classes have been presented independently of each other in the three parts, although we use and develop the linear programming relaxation techniques discussed before to obtain our results. We now elaborate about each of them and state our contributions.

1.2.1 Connectivity Problems

Given an undirected network with costs on edges, and a subset of vertices called terminals, the *minimum Steiner tree* problem is to find the cheapest sub-network which maintains the connectivity of the terminals. This is one of the classical NP-hard combinatorial optimization problems with a slew of applications ranging from computational biology to VLSI chip design to airport hub locations. The best known approximation algorithm for the minimum Steiner tree problem is a 1.55-factor algorithm due to Robins and Zelikovsky [86] while the best known hardness of approximation factor is 96/95 due to Chlebik and Chlebikova [23].

In this thesis, we propose a new *geometric* lower bound on the cost of the minimum weight Steiner tree. Informally, the lower bound corresponds to a linear function of a distance-preserving l_1 embedding of the vertices of the graph onto a k-dimensional λ-simplex[1]. We show that the *best* lower bound can be captured via a linear program, which we call the *simplex-embedding LP*. Moreover, we show that this LP is in some sense equivalent to the *bidirected cut LP relaxation* for the minimum Steiner tree.

The bidirected cut LP relaxation is a well-studied relaxation for the Steiner tree problem whose exact integrality gap is not known although the relaxation has been known for two decades. Our new relaxation in fact gives a geometric approach to attack the bidirected cut relaxation. We use this geometric framework to prove that for a large class of graphs, called *quasi-bipartite* graphs, the integrality gap of the bidirected cut relaxation is at most 4/3. The best known upper bound before our work was 3/2 due to Rajagopalan and Vazirani [84].

We also give algorithms for the Steiner tree problem on quasi-bipartite graphs. One of our algorithms, called Smart-Embed, runs in almost linear time to give a 3/2-factor algorithm and another runs in almost quadratic time to give a $\sqrt{2}$ factor algorithm. The algorithm which we use to prove the upper bound of 4/3 however runs in weakly polynomial time. All our algorithms are primal-dual algorithms. We should remark here that the best known approximation algorithm for quasi-bipartite graphs is also by Robins and Zelikovsky [86] obtaining a factor of 1.28 and runs in almost cubic time. We also remark that their algorithms are not LP-based.

Why is the bidirected cut relaxation for the Steiner tree problem important? First of all, the largest lower bound on the integrality gap known is only 8/7 (by Goemans[42] and Skutella [63]) and the true value could well be that, which would probably imply a better approximation algorithm for the Steiner tree problem. Secondly, the bidirected cut relaxation is a strengthening of another well-known relaxation for the Steiner tree problem called the *undirected cut* relaxation. The undirected cut relaxation has an integrality gap of 2 for the Steiner tree problem. More interestingly, the integrality gap is 2 even for generalizations of the Steiner tree problem (the Steiner forest problem and the Steiner network problem). One can thus hope for similar developments for these problems if progress is made on the bidirected cut relaxation for Steiner trees. No approximation factor better than 2 is known for these problems and the hardness is the same as that for the Steiner tree problem. That said, we remark that generalizations of the bidirected cut relaxation to the Steiner forest and network problems is not natural; and moreover for all we know the

[1] k is the number of terminals; a λ-simplex is the set of points with non-negative coordinates which add up to λ; a distance preserving embedding is such that the l_1-distance between the mapping of two vertices doesn't exceed twice the distance between them on the graph.

integrality gap of the bidirected cut relaxation for the Steiner tree problem could might as well be 2.

Details of the above can be found in Chapter 2. This is joint work with Nikhil Devanur and my advisor, Vijay V. Vazirani, and a preliminary version of the work appeared in [17].

1.2.2 Allocation Problems

In its generality, a resource allocation problem is to utilize a fixed amount of resources as effectively as possible, and such problems are ubiquitous in operations research, computer science and economics. In this thesis, we look at allocation problems motivated by the budgeted sponsored search auctions run by Internet search engine companies.

In a budgeted auction, bidders, who are the advertising companies in the sponsored search context, bid on a set of items, which are query keywords (like ``cheap hotels Atlanta'') and also specify a budget more than which they are unwilling to pay. Various issues arise in such auctions and have led to many algorithmic and economic questions. We discuss two such algorithmic questions in this thesis.

The first problem we study is the *maximum budgeted allocation* (MBA) problem: to allocate a set of indivisible goods to budgeted bidders so as to maximize the total revenue generated. Similar allocation problems have been studied in the literature, and in fact, a generalization of this problem called the *submodular welfare maximization* (SWM) problem has been extensively studied. In SWM, each agent i has a submodular utility function u_i, and the goal is to find an allocation of items maximizing the sum of utilities of the agents. It is not hard to see that MBA is a special case of SWM.

The best known approximation algorithm of MBA before our work was in fact the best algorithm for SWM due to Feige and Vondrák[35]. The factor was $1 - 1/e + \rho$ for some $\rho \leq 0.001$. Moreover, nothing better than NP-hardness was known for the problem. In this thesis, we improve in both directions. We give a 3/4-factor approximation algorithm and show that it is NP-hard to approximate MBA to a factor better than 15/16. Moreover, we use our hardness techniques to prove improved inapproximability results for many other allocation problems: (15/16 for SWM, 10/11 for GAP, 13/14 and 10/11 for node-weighted and edge-weighted maximum spanning star forest problem).

In fact, we give two approximation algorithms for MBA. Both of the algorithms are based on a natural LP relaxation for the problem. The first approximation algorithm is based on the *iterated rounding* schema of Jain[51] and is the first iterated rounding algorithm for a maximization problem. The second approximation algorithm is a primal-dual algorithm. Ours is one of the few primal-dual algorithms for a maximization problem — the primal-dual schema is normally tailored for minimization problems. Being primal-dual, our algorithm does not require solving the linear program and thus has significantly lower running time. However, the algorithm achieves only a factor of $3/4(1 - \epsilon)$ and runs in time $\tilde{O}(nm/\epsilon)$,

where n is the number of agents, m the number of items and the $\tilde{}$ hides logarithmic factors.

This work is presented in Chapter 3 and is joint work with Gagan Goel [18].

The second class of problems we study are *online* allocation problems when the set of items being allocated are not known. Rather, items arrive one-at-a-time and whenever an item arrives an irrevocable decision about its allocation has to be made. We consider various online allocation problems, the most general being online GAP and the most specific being the online knapsack problem.

Once more, the main motivation for studying these problems are the budgeted sponsored search auctions. In these auctions, the set of items which are queries on keywords are not known in advance and a decision of which advertisement to display has to be made in real time. Moreover, we also show that online allocation problems can also model bidding strategies of various budget-constrained bidders.

We use competitive analysis to measure the performance of our algorithms. We show that under certain necessary assumptions: a) weights of items being much smaller than the capacity of bins and b) the ratio of profit-to-weight ratio of items being bounded by L and U from below and above, our algorithms are $(\ln(U/L)+2)$ competitive for online-GAP and $(\ln(U/L)+1)$ competitive for the online-knapsack problem. We show that our algorithms are almost optimal by proving that *any* online algorithm for the online knapsack problem will have a competitive ratio of $(\ln(U/L)+1)$.

This work is presented in Chapter 4 and was started as an summer internship at HP Labs, Palo Alto, as part of the keyword-bidding project. In fact the bidding strategies implied by our algorithm (and modifications of it) were implemented and tested with real data obtained from the Internet. These implementations performed with a much better performance than those suggested by the theoretical result. In this thesis, we include only the theoretical portion. The project was conducted along with Dr. Yunhong Zhou and Dr. Rajan Lukose of HP Labs, Palo Alto, and parts of this work appeared in [20].

1.2.3 Design Problems

Consider the following question: Given an uncapacitated network, how should one distribute on the arcs a total capacity of one unit so as to maximize the minimum cut in the network? This question is what we call a typical *design* question and every optimization problem has a design version of it. In this thesis, we formalize this notion, and show that if the optimization version can be solved in polynomial time, then the design version can be solved in polynomial time as well. Moreover, if the optimization problem has an α-factor polytime approximation algorithm, then so does the design version. We first provide a proof using the ellipsoid method for solving linear programs. In the special case when costs are linear, we adapt algorithms from the learning theory literature to give faster algorithms for the design versions. More precisely, we adapt the multiplicative weights update algorithm of

Freund and Schapire [38] to solve zero-sum games to get fast algorithms for design problems.

Design versions of optimization problems are closely related to packing versions of the same. In the case when costs are linear, the max-min design version of a problem is dual to the fractional packing version of the problem. Using our framework, we study the fractional Steiner tree packing problem and derive old and new results about fractionally packing Steiner trees in a "black-box" fashion. Furthermore, we look at the special case of packing spanning trees, and give an alternate proof of the Nash-Williams-Tutte theorem about packing spanning trees.

This work is presented in Chapters 5 and 6. Parts of this appeared as joint work with Aranyak Mehta and Vijay V. Vazirani in [19].

CHAPTER II

A GEOMETRIC APPROACH TO THE STEINER TREE PROBLEM

The Steiner tree problem is one of the basic problems of combinatorial optimization: Given a weighted graph and a specified subset of vertices, find the smallest weight sub-graph in which the subset of vertices are connected. It is not hard to see such a smallest weight sub-graph is a tree.

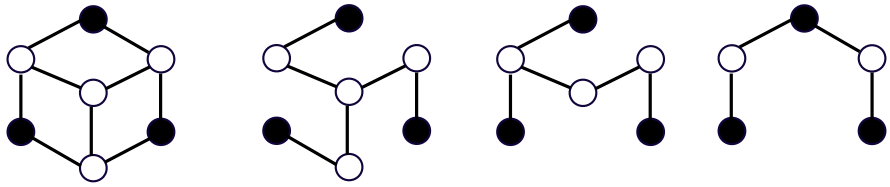

Figure 1: Examples of Steiner trees

In the left-most graph, the dark nodes form the subset of vertices to be connected. The three trees to the right are Steiner trees in decreasing order of number of edges.

The problem is fundamental in theoretical computer science with applications in various fields including VLSI design layout, hub-locations in telecommunication, phylogenetic trees in computational biology, and so on. Computationally however, it is among the classic 21 NP-complete problems of Karp [57] and the approximability of the problem is one of the most notable open questions in the approximation algorithm theory.

In this chapter, we investigate the approximability of the metric Steiner tree problem via LP-relaxations. We use geometry to develop a new way of lower bounding the cost of the optimal Steiner tree. The best such lower bound can be captured via an LP, which we call the *simplex-embedding LP*. A short description of this LP is that it is an l_1-embedding of the given metric on a simplex, maximizing a linear objective function. Interestingly enough, the dual of the simplex-embedding LP is a relaxation of the metric Steiner tree problem having the same integrality gap as the well-studied bidirected cut relaxation of Edmonds [31]; thus our new relaxation gives a new geometric way of studying the bidirected cut relaxation.

For a special class of graphs called quasi-bipartite graphs, we use the geometric approach to obtain dual-growing procedures and give three approximation algorithms for the metric Steiner tree problem on quasi-bipartite graphs. The first is a straightforward primal-dual algorithm achieving a factor of 3/2. We then modify the primal-dual schema via a new algorithmic idea to get a $\sqrt{2}$ and 4/3-factor respectively. The first two algorithms run in strongly polynomial time (almost linear and quadratic), while the one with the best factor

has an implementation only in weakly polynomial time.

Related Work: Historically, the idea of using an extra vertex to get a shorter tree connecting three points on the plane goes back to Torricelli and Fermat in the seventeenth century. The Euclidean Steiner tree problem, in its full generality, was first defined by Gauss in a letter to his student, Schumacher. This problem was made popular by the book of Courant and Robbins [29], who attributed the problem to the nineteenth century geometer, Jakob Steiner. The rich combinatorial structure of this problem was explored by many researchers; e.g., see the books by Hwang, Richards and Winter [48] and Ivanov and Tuzhilin [49].

The use of the bidirected cut relaxation for the Steiner tree problem goes back to Edmonds[31] who showed the relaxation is exact for the the case of spanning trees. Wong [95] gave a dual ascent algorithm for this relaxation, and Chopra and Rao[26, 27] studied the properties of facets of the polytope defined by the relaxation. Goemans and Myung[43] study various undirected equivalent relaxations to the bidirected cut relaxation.

The best approximation algorithm for the Steiner tree problem is due to Robins and Zelikovsky [86]. The authors prove a guarantee of 1.55 for general graphs and also show that when restricted to the quasi-bipartite case, the factor of their algorithm can be improved to 1.28. However, it is not clear if their results imply an upper bound on the integrality gap of the bidirected cut relaxation.

2.1 Preliminaries

Let $G = (V, E)$ be an undirected graph with edge costs $c : E \to \mathbb{R}$. $R \subseteq V$ denotes the set of *required* vertices. These are also called *terminals* and we use both terms interchangeably. The vertices in $S = V \setminus R$ are called *Steiner* vertices. The Steiner tree problem is to find the minimum cost tree connecting all the required vertices and some subset of Steiner vertices. We abuse notation and denote both the optimum tree and its cost as OPT. Also, given a set of vertices X, we denote the minimum spanning tree on X and its cost as $MST(X; c)$ or simply as $MST(X)$ when c is clear from the context. The edge costs can be extended to all pairs of vertices such that they satisfy triangle inequality (simply define the cost of (u, v) to be the cost of the shortest path from u to v). This version is called the metric Steiner tree problem. The two versions are equivalent.

Let $\mathcal{U} := \{U \subsetneq V : U \cap R \neq \varnothing \text{ and } U^c \cap R \neq \varnothing\}$ denote the subsets of V which contain at least one required vertex but not all. Let $\delta(U)$ denote the edges with exactly one end point in U. Note that any Steiner tree would use at least one edge in $\delta(U)$. This motivates the following *undirected cut relaxation* of the Steiner tree problem:

$$\min\{\sum_{e \in E} c(e)x_e : \quad x(\delta(U)) \geq 1, \ \forall U \in \mathcal{U}; \ x \geq 0\}$$

The MST on R is known [44] to be within factor 2 of the fractional optimum of this LP, so this relaxation has an integrality gap of at most 2. However, even in the case when G is just an n-cycle with each edge having cost 1, and $S = \varnothing$, that is, all vertices are required; the solution giving $x_e = 1/2$ on all edges is a valid solution of cost $n/2$. The minimum spanning tree is $n - 1$ and thus making n large, the integrality gap of the undirected cut relaxation can be arbitrarily close to 2, even for instances problem when there are no Steiner vertices.

2.1.1 The bidirected cut relaxation

Edmonds [31] introduced the following stronger relaxation called the *bidirected cut relaxation*[1]. Fix an arbitrary required vertex r as the root. Now replace each undirected edge (u, v) with two directed arcs $(\vec{u, v})$ and $(\vec{v, u})$, each of cost $c(u, v)$ (hence the name: bidirected). Call the set of arcs $\vec{E}$. The set of valid sets $\mathcal{U}$ are now those which contain the root but not all the required vertices. Thus $\mathcal{U} := \{U \subsetneq V : U \cap R \neq \varnothing \text{ and } r \notin U\}$. The observation now is that if the edges of any Steiner tree are directed to point away from the root, then at least one arc in this directed arborescence must be in the cut set $\delta^+(U)$ of arcs going out of U. This gives the bidirected cut relaxation for the minimum Steiner tree problem.

$$\min\{\sum_{e \in \vec{E}} c(e)x_{\vec{e}} : \quad x(\delta^+(U)) \geq 1, \quad \forall U \in \mathcal{U}; \quad x \geq 0\} \tag{BCR}$$

We denote the optimum of the above LP on a graph G as $BCR(G)$. Firstly, observe that the bidirected cut relaxation is stronger than the undirected cut relaxation. To see this note that any solution to LP(BCR) corresponds to a solution to the undirected cut relaxation: for every undirected edge add the variables on its two corresponding arcs. Edmonds [31] showed that the bidirected cut relaxation is exact for the MST problem, i.e., the integrality gap of the relaxation is 1 for spanning tree instances. In fact, in Section 2.3 we will prove a much stronger statement.

For Steiner trees however, no upper bound better than 2 (which is implied by the undirected cut relaxation) is known on the integrality gap of LP(BCR). Nevertheless, it is believed to be strictly better. Goemans [42] showed an example where the gap is $\frac{8}{7}$, which is the largest known. The only algorithmic results using the bidirected relaxation that we are aware of prior to our work are: a 6/5 factor algorithm for the class of graphs containing at most 3 required vertices [42] and a factor 3/2 algorithm by Rajagopalan and Vazirani [84] for the class of quasi-bipartite graphs, i.e., graphs that do not have edges connecting pairs of Steiner vertices.

[1] Actually, Edmonds only was concerned with the spanning tree problem, however the generalization to the Steiner tree problem is natural

A graph is called *quasi-bipartite* if there are no edges between any two Steiner vertices[2]. The class of quasi-bipartite graphs is a non-trivial class for the bidirected cut relaxation. In fact, recently Skutella (as reported by Könemann et.al.[63]) exhibited a quasi-bipartite graph on 15 vertices for which the integrality gap of the bidirected cut relaxation is 8/7. This ratio matches the erstwhile best known example on general graphs of Goemans[42] where the ratio was met as a limit. Moreover, the best known hardness results for the Steiner tree problem in this class of graphs is quite close to that known in general graphs ($\frac{128}{127}$ versus $\frac{96}{95}$)[23].

The bidirected cut relaxation has interesting structural properties which have been exploited in diverse settings. [53] use this LP for giving a factor 2 budget-balanced group-strategy-proof cost-sharing method for the Steiner tree game. Let α denote the integrality gap of this relaxation. Agarwal and Charikar [1] (see also Chapter 6) prove that for the problem of multi-casting in undirected networks, the coding gain achievable using network coding is precisely equal to α. The latter result holds when restricted to quasi-bipartite networks as well. Consequently, for these networks, the previous best bound was 3/2 [84], and our result improves it to 4/3.

Organization: In Section (2.2) we show the geometric theorem giving the lower bound, and other results relevant to it. In Section(2.3), we demonstrate a dual-growing embedding procedure for quasi-bipartite graphs. In Sections (2.4), (2.5) and (2.6) we give our 3/2, $\sqrt{2}$ and $\frac{4}{3}$ factor primal-dual approximation algorithms respectively.

2.2 A Geometric Lower Bound and its Consequences

We first present a special case of the geometric theorem, for the sake of ease of presentation. Let Δ_k be the unit simplex in $\mathbb{R}^k$, that is, $\Delta_k := \{x \in \mathbb{R}^k : \sum_{i\in[k]} x(i) = 1\}$, where $x(i)$ is the ith coordinate of x. The corners of Δ_k are the unit vectors in $\mathbb{R}^k$. Let T be any Steiner tree in Δ_k connecting the corners, that is, T is a tree whose vertices are the corners and any number of finitely many other points in Δ_k. That is, T is a finite tree whose vertices are picked from Δ_k and contains the corners as a subset. For any point v in T, let $deg_T(v)$ denote the degree of the vertex in the tree T. Define the distance between two points to be half the l_1-distance, also called the *variational distance*; for any two points $x, y \in \Delta_k$, $d(x, y) := \frac{1}{2}\sum_{i=1}^{k} |x(i) - y(i)|$. (The half is so that two corners are at a distance of 1). For any edge $e = (x, y)$, let $d(e) := d(x, y)$. Let $d(T) := \sum_{e\in T} d(e)$. Then

Theorem 2.2.1 $d(T) \geq k - 1$.

[2]This might seem to contradict the assumption that the graph can be assumed to have all edges present. To be more precise the the cost of an edge between any two Steiner vertices equals the cost of the shortest path between the two and no two Steiner vertices are adjacent in the path.

Note that if the vertices in T were only the corners, that is T was a spanning tree, then the relation holds with equality since any two corners are at a distance of 1. The theorem says that any other Steiner points from the simplex don't improve upon the MST, w.r.t $d()$. This is somewhat counter-intuitive, since in most geometric spaces, the Steiner points do improve upon the MST. What is special here is the l_1-distance, and the location of the points on the simplex.

Proof: Let $R = \{e_1, e_2, \ldots, e_k\}$ be the unit vectors in $\mathbb{R}^k$. The proof follows by a careful[3] counting argument. First of all, we get rid of the absolute values that occur in the l_1 distance. For every edge (x, y) in T, and $i \in [k]$, note that either x is on the unique path from y to e_i or y is on the unique path from x to e_i, depending on who is *nearer* to e_i in T. If x is on the unique path from y to e_i in T, then we lower bound $|x(i) - y(i)|$ by $x(i) - y(i)$ (and vice versa).

$$d(T) = \frac{1}{2} \sum_{(x,y)\in E} \sum_{i\in[k]} |x(i) - y(i)| \quad \geq \frac{1}{2} \sum_{(x,y)\in E} \sum_{i\in[k]} sgn_i(x, y) \cdot (x(i) - y(i)),$$

where $sgn_i(x, y)$ is 1 if x is nearer to e_i than y in T, and -1 otherwise. Thus $d(T)$ is lower bounded by a linear combination of the $x(i)$'s for every vertex x in T, and every coordinate i.

Note that in the summation, $x(i)$ occurs with a positive sign for each of its neighbor in T which is farther from e_i than itself. Thus, the coefficient of $x(i)$ is $\frac{1}{2}(deg_T(x) - 2)$, except for $e_i(i)$, whose coefficient is $\frac{1}{2} deg_T(e_i)$. Therefore,

$$\begin{aligned} d(T) &\geq \frac{1}{2} \sum_{x\in T, i\in[k]} (deg_T(x) - 2)\, x(i) + \frac{1}{2} \sum_{i\in[k]} 2e_i(i) \\ &= \frac{1}{2} \sum_{x\in T} (deg_T(x) - 2) + \sum_{i\in[k]} 1 \\ &= k - 1. \end{aligned}$$

where the equality in the second line holds because $\sum_{i\in[k]} x(i) = 1$ and the last equality follows from the fact that in a tree $\sum_{x\in T} deg(x) = 2|T| - 2$. □

The general theorem allows two concessions on the location of the points: first, the points need not be on the unit simplex, all points are in the λ-simplex $\Delta_k^{(\lambda)}$, defined as $\{x \in \mathbb{R}^k : \sum_{i\in[k]} x(i) = \lambda\}$, for some parameter $\lambda > 0$. The second is that the required points need not be at the corners of the simplex. In particular, let $z_1, z_2, \cdots, z_k$ be any k points in $\Delta_k^{(\lambda)}$ and T be any tree connecting these points using any finite number of other points in $\Delta_k^{(\lambda)}$. Let $d()$ be the variational distance defined above. Then,

Theorem 2.2.2 $d(T) \geq \gamma(z) := (\sum_{i\in[k]} z_i(i) - \lambda)$.

[3] An easy counting argument shows that $d(T) \geq \frac{1}{2}(k - 1)$.

Note that when the z_i's are corners of a unit simplex, the above implies Theorem (2.2.1).

Proof: Let $e_1, \cdots, e_k$ now be the corners of the λ-simplex. From T, construct the tree T' connecting the point z_i to e_i. Note that the distance between z_i and e_i is $d(z_i, e_i) = \lambda - z_i(i)$. Theorem (2.2.1) can be easily modified to give $d(T') \geq (k-1) \cdot \lambda$. Thus, $d(T) = d(T') - \sum_{i \in [k]} (\lambda - z_i(i)) \geq (k-1) \cdot \lambda - \sum_{i \in [k]} (\lambda - z_i(i)) = \gamma(z)$. □

Theorem (2.2.2) can be used to get a lower bound on the minimum Steiner tree in a graph $G = (R \cup S, E)$ as follows. Given G, suppose $|R| = k$ and $R = [k]$. Embed the vertices of the graph onto $z : V \to \Delta_k^{(\lambda)}$, and call an embedding *valid* if for all edges $(u, v) \in E$, $c(u, v) \geq d(z_u, z_v)$. Henceforth, we will abuse notation and denote $d(z_u, z_v)$ as $d(u, v)$. Now given any valid embedding z, for any Steiner tree T of G, $c(T) \geq d(T) \geq \gamma(z)$, where $d(T)$ is the length of the tree formed by the embedded points. In particular, we have

Theorem 2.2.3 *If z is a valid embedding of G, then $OPT \geq \gamma(z)$.*

2.2.1 Connections to the bidirected cut relaxation

For any embedding of vertices, Theorem (2.2.3) provides a lower bound on the minimum Steiner tree in the graph. Thus the best lower bound is given by $\max \{\gamma(z) : z \text{ is valid}\}$. The above maximum can be obtained by solving the following linear program which we call the *simplex-embedding LP*.

$$\begin{aligned} \max \ \{\gamma(z) = \sum_{i \in [k]} z_i(i) - \lambda : & \qquad \text{(SimpEmb)} \\ \sum_{i \in k} z_v(i) = \lambda, \ \forall v \in V; & \\ z_v(i) - z_u(i) \leq d_i(u, v) \ \forall i \in [k], (u, v) \in E; & \\ z_u(i) - z_v(i) \leq d_i(u, v), \ \forall i \in [k], (uv) \in E; & \\ \frac{1}{2} \sum_{i \in [k]} d_i(u, v) \leq c(u, v), \ \forall (u, v) \in E; & \\ z_v(i), d_i(u, v) \geq 0, \ \forall v \in V, i \in [k], (u, v) \in E\} & \end{aligned}$$

Given input graph G, let $SE(G)$ denote the maximum of the above LP. In the next theorem we prove that for any graph G and cost vectors c, $SE(G) = BCR(G)$. That is, the lower bound obtained by the above LP and the bidirected cut relaxation is the same. The proof uses the dual of the simplex-embedding LP which is as follows (after a scaling step).

$$SE(G) = \text{minimize } \{ \sum_{(u,v)\in E} c(u,v)x_{uv} : \qquad \text{(SE-Dual)}$$

$$x_{uv} \geq f_i(uv) + f_i(vu),\ \forall i \in [k],\ (u,v) \in E;$$

$$\sum_{v:(u,v)\in E} (f_i(uv) - f_i(vu)) \geq \alpha(v),\ \forall i \in [k], v \in V \setminus i;$$

$$\sum_{v:(i,v)\in E} (f_i(iv) - f_i(vi)) \geq \alpha(i) + 2, \qquad \forall i \in [k];$$

$$\sum_{v} \alpha(v) = -2;$$

$$f_i(uv), f_i(vu), x_{uv} \geq 0, \quad \forall (u,v) \in E, i \in [k]\}.$$

To interpret LP(SE-Dual), one can think of x_{uv} as capacity of edge (u,v). There are k circulations — f_i for every required vertex i each satisfying the capacity constraint and moreover, the supplies (excess flows) for f_i at every vertex v is $\alpha(v)$ (it could be negative) except at the vertex i, where it is $\alpha(i) + 2$. The last equality constraint implies the total supplies sum up to zero, as it should be in a circulation.

We now show directly that LP(SE-Dual) is a relaxation of the minimum Steiner tree problem thus giving a direct proof of Theorem (2.2.3). Given any Steiner tree T, consider the following solution to LP(SE-Dual): $x_{uv} = 1$ for all $(u,v) \in E(T)$, and 0 for all other edges; $\alpha(v) = deg_T(v) - 2$ for all $v \in V(T)$ and 0 for all other vertices. Note that $\sum_v \alpha(v) = -2$ and $\sum_{(u,v)\in E} c(u,v)x_{uv}$ is the cost of the tree. It remains to define the circulations. For required vertex i, consider the unique out-tree rooted at i formed by directing all edges of T away from i. For every edge (u,v) in T, let $f_i(uv)$ or $f_i(vu)$ be 1 depending on the direction of the arc in the out-tree. All other arcs carry 0 circulation. Note that for any vertex v in the tree other than i, the total flow coming in is 1, and flow going out is $deg_T(v) - 1$; while for i, the total flow going out is $deg_T(i)$. Thus the circulation satisfies the supply constraints. This shows that LP(SE-Dual) is a relaxation of the minimum Steiner tree problem.

The following theorem shows the equivalence between LP(SE-Dual) and LP(BCR). In [43], Goemans and Myung provide two vertex weighted relaxations which are equivalent to the bidirected cut relaxation. Although our relaxation is different, our proof of equivalence follows on similar lines.

Theorem 2.2.4 *Given any graph G, $SE(G) = BCR(G)$.*

Proof: Theorem (2.2.4) follows by showing there exists a feasible solution to LP (BCR) of value Γ if and only if there exists a feasible solution to LP (SE-Dual) of value Γ. Note that LP(BCR) is the following:

$$\min\{\sum_{e\in\vec{E}} c(e)x_{\vec{e}}: \quad x(\delta^+(U)) \geq 1, \quad \forall U \in \mathcal{U}; \quad x \geq 0\}$$

LP (BCR) $\geq$ **LP** (SE-Dual)**:** Let $\{y_{(\vec{u,v})}\}_{(\vec{u,v})\in\vec{E}}$ be a feasible solution of LP (BCR) of cost Γ with root r. The corresponding solution to LP (SE-Dual) is as follows: $x_{uv} := y_{(\vec{u,v})} + y_{(\vec{v,u})}$ and $\alpha(v)$ is the *supply* at vertex v which is the difference of the outgoing $y_{(\vec{v,u})}$'s and the incoming $y_{(\vec{u,v})}$'s, except at r where $\alpha(r)$ is the supply-2. This ensures the sum of α's is -2. What remains is to describe the flows. The flow corresponding to r just mimics the $y_{(\vec{u,v})}$'s. That is $f_r(uv) := y_{(\vec{u,v})}$ and $f_r(vu) := y_{(\vec{v,u})}$, for all edges (u, v). It is easy to see that every constraint is till now satisfied. To get the flows corresponding to another required vertex j, we use the fact that the *minimum r-j cut* in the digraph with arc set $\vec{E}$ and capacities $y_{(\vec{u,v})}$'s is at least 1 since the solution is feasible for LP (BCR). This implies there is a standard flow g_{rj} from r to j in this digraph. The flow f_j is found by *subtracting* $2g_{rj}$ from f_r.

To be precise, for each edge (u, v) we set

$$f_j(uv) := \max[0, f_r(uv) - 2g_{rj}((\vec{u,v}))] + \max[0, 2g_{rj}((\vec{v,u})) - f_r(vu)]$$

and

$$f_j(vu) := \max[0, f_r(vu) - 2g_{rj}((\vec{v,u}))] + \max[0, 2g_{rj}((\vec{u,v})) - f_r(uv)]$$

It is easy to check that f_j satisfies the constraints of LP(SE-Dual). Since g_{rj} increases the supply of j by 2 and decreases that of i by 2, the supply constraints are guaranteed. The capacity constraints are guaranteed by noting

$$f_j(uv) + f_j(vu) \leq \max[(f_r(uv) + f_r(vu)), (f_r(uv) - f_r(vu) + 2g_{rj}((\vec{v,u})))]$$

both of which are less than $y_{(\vec{u,v})} + y_{(\vec{v,u})} = x_{uv}$.

Thus the solution is feasible for LP(SE-Dual) and is of value Γ.

LP(SE-Dual) $\geq$ **LP**(BCR): Let $(\{x\}, \{f_i\}, \{\alpha\})$ (respectively over edges, arcs and vertices) be a solution to LP (SE-Dual). WLOG by adding circulations if necessary, we can assume for all edges (u, v), and i we have $x_{uv} = f_i(uv) + f_i(vu)$. For LP(BCR), let r be the chosen root. Then the solution is: for all edges (u, v), $y_{(\vec{u,v})} := f_r(uv)$ and $y_{(\vec{v,u})} := f_r(vu)$ To see feasibility for LP (BCR), we must show across any cut S separating r and a required vertex j, we have $\sum_{(\vec{u,v})\in\delta^+(S)} f_r(uv) \geq 1$. To see this, consider the flow g_{rj}: the difference between f_r and f_j. To be precise:

$$g_{rj}((\vec{u,v})) = \max[0, f_r(uv) - f_j(uv)] + \max[0, f_j(vu) - f_r(vu)]$$

Observe that the supply of the flow g_{rj} on every vertex other than r or j is 0 and on r is 2 and j is -2. This is because the supplies of f_r and f_j match on every vertex other than

r and j where they differ by 2. Thus it is a standard flow from r to j of value 2, implying across any cut S as above, $\sum_{(\vec{u,v})\in\delta^+(S)} g_{rj}((\vec{u,v})) \geq 2$. The proof ends by noting for any arc $(\vec{u,v})$, $g_{rj}((\vec{u,v})) \leq 2f_r(uv)$, because each term in the above definition of g_{rj}, is less than $f_r(uv)$. The second term is less since $f_r(uv) + f_r(vu) = x_{uv} \geq f_j(vu)$. □

What the above theorem shows is that the LP(SimpEmb) can be interpreted as a geometric dual to the bidirected cut relaxation. In Section 2.3 we explain how we design algorithms using the primal-dual schema with this new geometric dual.

2.2.2 A stronger LP relaxation

Note that in LP(SE-Dual), the $\alpha(v)$'s are free variables. This is because the interpretation of $\alpha(v)$ as $deg_T(v) - 2$ implies that for leaves $\alpha(v)$ could be negative. However, in any optimal Steiner tree, no Steiner vertex would be a leaf. Therefore, we could add the constraint $(\alpha(v) \geq 0,\ \forall v \in S)$, and the new relaxation, call it LP(SE-Dual2), would still be a feasible relaxation for the minimum Steiner tree problem.

$$\text{minimize } \{ \sum_{(u,v)\in E} c(u,v)x_{uv} : \tag{SE-Dual2}$$

$$x_{uv} \geq f_i(uv) + f_i(vu),\ \forall i \in [k],\ (u,v) \in E;$$

$$\sum_{v:(u,v)\in E} (f_i(uv) - f_i(vu)) \geq \alpha(v),\ \forall i \in [k], v \in V \setminus i;$$

$$\sum_{v:(i,v)\in E} (f_i(iv) - f_i(vi)) \geq \alpha(i) + 2, \qquad \forall i \in [k];$$

$$\sum_{v} \alpha(v) = -2;$$

$$f_i(uv), f_i(vu), x_{uv} \geq 0, \quad \forall (u,v) \in E, i \in [k];$$

$$\alpha(v) \geq 0, \quad \forall v \in S\}. \tag{4}$$

The corresponding change in the simplex-embedding LP is that for the Steiner vertices, the equality constraint $\sum_{i\in[k]} z_v(i) = \lambda$ is relaxed to an inequality constraint $\sum_{i\in[k]} z_v(i) \geq \lambda$. Call this new LP, LP(SimpEmb2). In other words, the Steiner vertices are allowed to embed "above" the simplex. It is not too hard to extend Theorem(2.2.2) to capture this as well.

It is clear that the optimum value of LP(SimpEmb2) is at least that of LP(SimpEmb) and thus it gives as good a lower bound. In fact, the following example in Figure(2.2.2) shows that on some instances the former is strictly better, and thus LP(SE-Dual2) is strictly a better relaxation than LP(SE-Dual) and also by Theorem(2.2.4), better than LP(BCR).

Nevertheless, the worst integrality gap known for LP(SE-Dual2) is also 8/7. Moreover, in the remainder of the paper we will investigate LP(SimpEmb) and thus all our results will

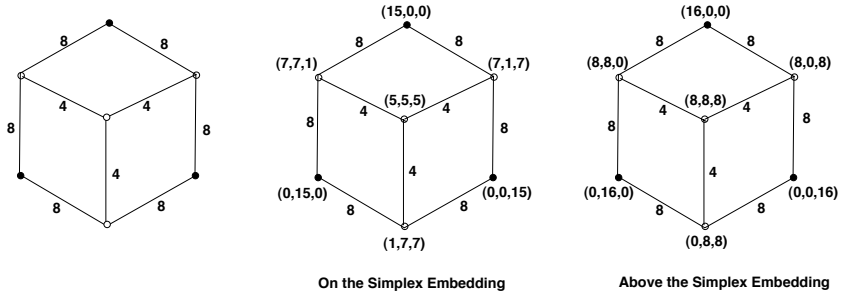

Figure 2: Integrality gap example for bidirected cut relaxation

Integrality gap of the bidirected cut relaxation for the graph is known to be 16/15 (due to Goemans). The middle figure shows an embedding on the simplex attaining a value of 15. The figure to the right shows how we can get a higher value if we allow Steiner vertices to move above the simplex. Note that the Steiner vertex at the center is not on the 16-simplex.

imply results for the bidirected cut relaxation.

2.2.3 Primal-Dual schema with the geometric dual

All the algorithms we give for the minimum Steiner tree problem fall in the following primal-dual schema: we construct a Steiner tree T and a feasible embedding z of the vertices of the graph onto a λ-simplex; moreover we prove $c(T) \leq \rho \cdot \gamma(z)$, and thus get a ρ-approximation for the Steiner tree problem as well as an upper bound on the integrality gap of LP(SE-Dual) and LP(BCR).

In the remainder of the paper we restrict ourselves to the special class of graphs called *quasi-bipartite graphs*. Such graphs do not have Steiner-Steiner edges. The best known upper bound on the integrality gap of LP(BCR) on such graphs was 3/2[84].

For such graphs, we first describe the dual growing procedure, EMBED, which gives us a feasible embedding of vertices. EMBED has the property that on quasi-bipartite graphs, if the MST on the required vertices is optimal, then it returns an embedding z equalling the cost of the MST on terminals. Otherwise, it returns a Steiner vertex v which helps the MST on the terminals.

The EMBED algorithm stops after returning the first Steiner vertex. In fact, one can extend EMBED, however a natural extension doesn't give good enough dual (the dual obtained can be as small as 1/2 of the cost of the optimal tree). In Section 2.4, we show how to modify EMBED and give a fast and simple primal-dual algorithm achieving the upper bound of 3/2 for quasi-bipartite graphs. The algorithm runs in near linear time compared to the previous weakly-polynomial time algorithm of Rizzi [85].

In Sections 2.5 and 2.6, we use EMBED *in rounds* (with a processing step before each round) to get $\sqrt{2}$ and $\frac{4}{3}$ factor approximations respectively for quasi-bipartite graphs. The $\sqrt{2}$ algorithm runs in strongly polynomial time, while the 4/3 algorithm takes the same

time as Rizzi's algorithm.

2.3 The EMBED algorithm

In this section, we describe the dual growing procedure EMBED which given a quasi-bipartite graph G and a cost function c does the following.

Case 1: If $MST(R)$ is the optimal Steiner tree, then it returns a feasible embedding z such that $\gamma(z) = MST(R)$. Note, in this case, $MST(R) = OPT = BCR$, since $MST(R) \geq OPT \geq BCR \geq \gamma(z)$.

Case 2: Or, returns a Steiner vertex v whose addition *strictly* helps the MST on R, that is, $MST(R \cup v) < MST(R)$. We say that EMBED *crystallizes* v.

The following theorem is immediate.

Theorem 2.3.1 *Given a quasi-bipartite instance G, if the addition of no Steiner vertex reduces the cost of $MST(R)$, then $MST(R) = BCR(G)$. In particular the integrality gap for the instance is* 1.

Note that the theorem implies the following important property about the bidirected cut relaxation for quasi-bipartite graphs: If the minimum spanning tree is optimal, then the relaxation is exact. We are now ready to describe EMBED.

The following is a continuous description of the algorithm, which can be easily discretized. The algorithm has a notion of time. It starts at time $t = 0$ and time increases at unit rate. At any time t, all required vertices are on the t-simplex, all Steiner vertices are below the t-simplex (sum of coordinates is less than t). The algorithm maintains a set of terminal-terminal edges T, which form a forest at any time t. Let K denote a connected component of required vertices formed with the edges of T. At time $t = 0$, the algorithm starts with $T = \varnothing$ and the components are singleton required vertices. All vertices start at the origin at $t = 0$. Before describing how the embedding at time t is formed, we make a few definitions.

Definition 1 *At any point of time, $d(u, v)$ is the half-l_1 distance between z_u and z_v, that is, $d(u, v) := \frac{1}{2} \sum_i |z_u(i) - z_v(i)|$. Another useful notation would be $d^+(\{u, v\}) := \sum_{i \in S_u} (z_u(i) - z_v(i))$, where S_u are the coordinates in which u dominates v. That is, $S_u := \{i : z_u(i) \geq z_v(i)\}$. Note that d^+ takes an ordered pair of vertices. Also observe that if z_u and z_v are both on the simplex, then $d^+(\{u, v\}) = d^+(\{v, u\}) = d(u, v)$.*

Definition 2 *Given a vertex v and a component K, let $d(v, K) := \min_{u \in K} d(u, v)$. Similarly define $d^+(\{v, K\})$ and $d^+(\{K, v\})$. For any two components $K, L \in \mathcal{K}$, let $c(K, L) := \min_{u \in K, v \in L} c(u, v)$. An edge (u, v) is* tight *if $d(u, v) = c(u, v)$.*

Definition 3 *A Steiner vertex v* links *to a component K if there exists $i \in K$ so that $d^+(\{i,v\}) = c(i,v)$. The edge (i,v) is called a link of v to K.*

Algorithm 1 EMBED

Required vertices: For each component K and required vertex $i \in K$, the algorithm increases the jth coordinate of i at rate $1/|K|$, for each $j \in K$. Clearly, this will keep required vertex i on the t-simplex. When an edge (i,j) goes tight, the algorithm merges the components containing i and j and adds (i,j) to T. It is instructive to note that when restricted to only required vertices, this actually mimics Kruskal's MST algorithm.

Steiner vertices: For each Steiner vertex v, the algorithm increases its coordinates depending on the components it has linked to. For each component K that v is linked to, the coordinates of v corresponding to K increase at rate $1/|K|$. Thus, henceforth $d^+(\{i,v\})$ remains the same for all terminals $i \in K$. Note that at $t = 0$, the Steiner vertex is linked to no component and remains at the origin till $t = c(i,v)$, where i is the closest terminal to v.

The algorithm terminates if the number of components becomes 1 (Case 1) or a Steiner vertex v hits the simplex (Case 2). In Case 1, the algorithm runs the following projection step.

Projection Step: If Case 1 happens at time $t = \lambda$, for every Steiner vertex v and coordinate j, $z_v(j) := z_v(j)\frac{\lambda}{\sum_i z_v(i)}$. The coordinates of the required vertices are kept the same.

The embedding procedure is described in Algorithm 1. The example in Figure 2.3 illustrates the algorithm on a graph with three required vertices.

We now show Algorithm 1 satisfies the conditions mentioned at the beginning of the section. In Case 1, the algorithm returns a tree T and embedding z. The way the required vertices are moved, it is clear that no terminal-terminal edge is over tight. To see that terminal-Steiner edges are not over tight, note that before the projection step, we have for every terminal i and every Steiner vertex v, $d^+(\{i,v\}) \leq c(i,v)$ (otherwise v links to the component containing i). After projection step, the coordinates of the Steiner vertex only increase, which means $d^+(\{i,v\})$ only decreases. Moreover, since v is on the simplex, we have $d^+(\{i,v\}) = d(i,v)$ and thus $d(i,v) \leq c(i,v)$.

We now need to show that tree T has cost $\gamma(z)$. In fact we prove something stronger. Given any connected component K, denote the restriction of T to K as $T[K]$.

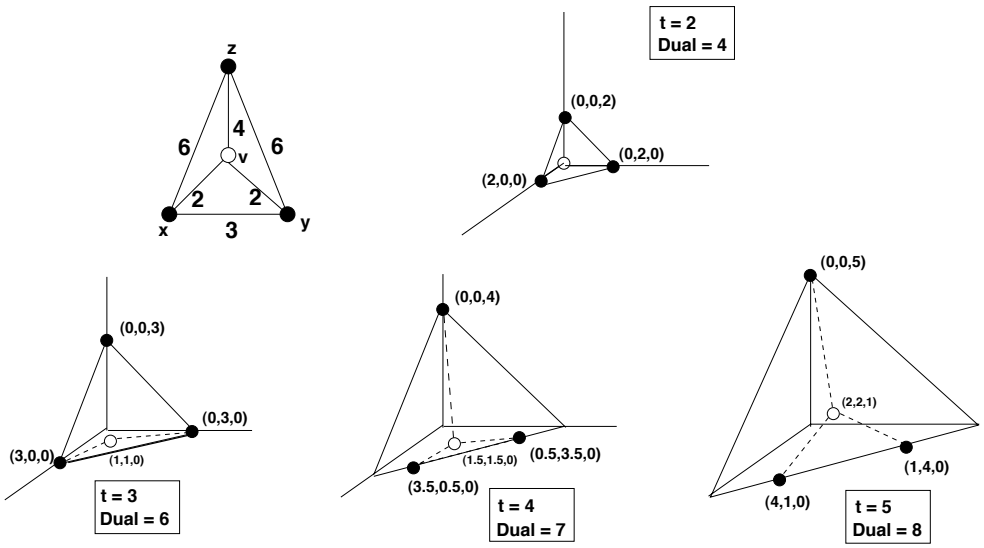

Figure 3: Snapshots of the running of EMBED on the graph above at times $t = 2, 3, 4, 5$.

Snapshots of the running of EMBED on the graph above at times $t = 2, 3, 4, 5$. At time $t = 2$, the Steiner vertex v links to the required vertices x and y, and increases its x and y coordinates at rate 1. At time $t = 3$, x, y merge. The edge (x, y) goes into $Remove(v)$. At time $t = 4$, v links to z, and moves in the zth coordinate as well. At $t = 5$, it hits the 5-simplex, terminating the algorithm. The tree shown with dotted lines pays exactly for the dual and is cheaper than the MST.

Lemma 2.3.2 *At any instant of time t, for any connected component K,* $c(T[K]) = \sum_{i \in K} z_i(i) - t$.

Proof: At time $t = 0$, the lemma holds vacuously. Since the quantity $\sum_{i \in K} z_i(i)$ increases at the same rate as time, we need to prove the lemma only in the time instants when two components merge. Suppose K, K' merge at time instant t due to edge (i, j) which comes in the tree, with $i \in K, j \in K'$. Note $d(i, j) = c(i, j) = t$. So for the new connected component $K \cup K'$, $\sum_{i \in K \cup K'} z_i(i) - t = \sum_{i \in K} z_i(i) - t + \sum_{i \in K'} z_i(i) - t + t = c(T[K]) + c(T[K']) + c(i, j) = c(T[K \cup K'])$. $\square$

In Case 2, when v hits the simplex, the algorithm returns v as the Steiner vertex helping the minimum spanning tree. In fact, we show that if v is linked to $K_1, \cdots, K_r$ when it hits the simplex, then v helps the MST of the required vertices in $P = \bigcup_l K_l$. This suffices since $\bigcup_l T[K_l]$ can be extended to an MST of R.

With each Steiner vertex v, we associate a subset of edges $Remove(v)$ of T. Suppose v is linked to K and K' and these merge at time t, due to edge (i, j), $i \in K$ and $j \in K'$. At this point, (i, j) is added to the set $Remove(v)$. Thus, a Steiner vertex may have more than one link into the same component, but for each extra link, there is an edge in $Remove(v)$. Let T_v be the tree formed by adding all the links incident at v to $\bigcup_l T[K_l]$ and deleting $Remove(v)$. The proof of the following lemma is very similar to that of Lemma (2.3.2).

Lemma 2.3.3 *At any instant of time,* $c(T_v) = \sum_{i \in P} z_i(i) - \sum_{i \in P} z_v(i)$.

Proof: At time $t = 0$, the lemma holds vacuously. Since the quantity $\sum_{i \in P} z_v(i)$ increases precisely at the rate $\sum_{i \in P} z_i(i)$, we need only check the lemma in when v links to a *new* component K. Suppose this happens at time t. This means, there is a terminal $j \in K$ with $c(v, j) = t$. Note that $\sum_{i \in K} z_i(i) = t$, by the way required vertices move. Thus the increase in both left-hand side and right-hand side is t, and thus equality holds. □

Hence when v hits the simplex, $\sum_{i \in P} z_v(i) = t$, and so $c(T_v) = \sum_{i \in P} z_i(i) - t < MST(P)$. Thus, we have proved the following theorem.

Theorem 2.3.4 *Given a quasi-bipartite graph G, the algorithm* EMBED *either returns a terminal spanning tree T and feasible embedding z with $c(T) = MST(R) = \gamma(z)$; or returns a Steiner vertex v with $MST(R \cup v) < MST(R)$.*

Remark: Note that the above algorithm and analysis do not use the fact the cost satisfies triangle inequality. We would need this for our algorithms in Sections 2.5 and 2.6 to work.

2.4 A $\frac{3}{2}$-factor approximation algorithm

The dual growing procedure EMBED suggests the following primal-dual algorithm: At each step, maintain the connected components of T; when a Steiner vertex v hits the simplex, *merge* all the components v was linked to by adding v and the various links connected to it to T and continue the dual growing procedure.

However, as the example in Figure(2.4) suggests, such an algorithm cannot give anything better than a factor 2.

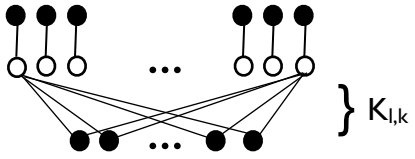

Figure 4: Example showing drawback of EMBED

In the above graph, the black vertices are terminals and the white vertices are Steiner. There are l Steiner vertices connected to k terminals via a complete bipartite graph $K_{l,k}$. Think of $l >> k >> 1$. There are l other terminals forming a perfect matching with the l Steiner vertices. Each edge is of length 1. Note that the optimum Steiner tree costs at least $2l$. In the procedure EMBED, all the Steiner vertices hit the t-simplex simultaneously at time $t = 1 + \frac{1}{k}$. Subsequently, no more dual can be grown and thus the total dual obtained is around $k + l$.

The above example suggests a shortcoming of the EMBED procedure: it grows the coordinates of all the Steiner vertices, although in any tree, at most one Steiner vertex is *useful*. To be precise, the k vertices at the bottom increases the coordinates of all the l

Steiner vertices at a very high rate (at rate k), although only one of them is useful for connecting the k terminals.

The main idea behind the 3/2-factor algorithm is it uses EMBED to recognize *useful* Steiner vertices *early* (not wait till they hit the simplex) and on recognizing merge the components linked to the vertex so that these components do not raise the coordinates of other Steiner vertices.

In short, the algorithm runs EMBED until some Steiner vertex is linked to three distinct connected components. If that is the case, it merges all the components, and adds the Steiner vertex and the three links to the tree. The idea is that this Steiner vertex should be helpful. If some Steiner vertex now hits the simplex, it must do so being connected to only *two* components. In this case, the algorithm merges these two components, and adds the Steiner vertex and the two links to the tree, and continues. The algorithm terminates when there is a single component. We state the algorithm formally in Algorithm 2. We use the definitions 1,2 and 3 from Section 2.3.

The algorithm maintains a set of edges and vertices, T, which is initialized to $E(T) = \varnothing$ and $V(T) = R$. At every time interval t, the algorithm maintains a family of connected components $\mathcal{K}$ of T; and an embedding $z : V \to \mathbb{R}^k_+$ of the vertices. At time $t = 0$, all the vertices of the graph are embedded at the origin. Time increases at rate 1.

Algorithm 2 SMART-EMBED

Until a Merge Step happens or the number of components become 1 do:

Component Vertices: For each component $K \in \mathcal{K}$, and for each terminal $i \in K \cap R$, the algorithm increases the ith coordinate of every vertex $v \in K$ (required or Steiner) at rate $1/|K \cap R|$.

Other Vertices: For each Steiner vertex v not in any component, the algorithm increases its coordinates depending on the components it has linked to. For each component K that v is linked to, the coordinates of v corresponding to K increase at rate $1/|K \cap R|$.

Merge Step: A Merge Step happens when one of the three events take place:

1. For some two components $K, L \in \mathcal{K}$, we have $c(K, L) = t$. In this case, we merge K and L into one component. All Steiner vertices linked to either K or L now link to the new component. Moreover, if a Steiner vertex is linked to both K and L, the costlier of its links to K and L is removed. This maintains that a Steiner vertex has only one link to a component. Let (u, v) be the edge which achieves $c(K, L)$. Add (u, v) to T.

2. Some Steiner vertex v not in any component hits the t-simplex, that is, $\sum_{i \in [k]} z_v(i) = t$. In this case, merge v and all the components it links to into one single component. For each component K, v links to, add the cheapest links of v to K to T. Add v to T as well.

3. Some Steiner vertex v not in any component links to three components K_i, K_j, K_l. In this case, merge v and K_i, K_j, K_l into one component. Once more all Steiner vertices linked to either K_i,K_j or K_l now link to the new component. Steiner vertices which link to two out of the three, discard their costlier link, as in Step 1. Add v and its links to K_i, K_j, K_l to T. Also project v onto the t-simplex as in EMBED.

It is clear that the algorithm returns a Steiner tree T in the end. Moreover, the feasibility of the embedding returned by EMBED also implies the feasibility of the embedding z returned by SMART-EMBED. We finish the section with the following theorem which shows that the tree is within 3/2 of the optimal.

Theorem 2.4.1 *The algorithm* SMART-EMBED *returns a tree T and a feasible embedding z, with $c(T) \leq \frac{3}{2}\gamma(z)$.*

Proof:

We show that at any time t, for any connected component K we have $c(T[K]) \leq \frac{3}{2}(\sum_{i \in K \cap R} z_i(i) - t)$. Henceforth, we abuse notation and denote $c(T[K])$ by $c(K)$. The proof is by induction

on t. At $t = 0$, the inequality holds vacuously. Moreover it is enough to check that the inequality is preserved after every merge step as between merge steps both sides of the inequality remain unchanged.

Suppose at time t, a merging of type 1 occurs: components K and L merge into one component, and the edge (u, v) is added to T. By induction we have $c(K) \leq \frac{3}{2}(\sum_{i \in K \cap R} z_i(i) - t)$ and $c(L) \leq \frac{3}{2}(\sum_{i \in L \cap R} z_i(i) - t)$. The cost of the new component $(K \cup L)$ increases by the cost of the edge (u, v). Note that by definition, $c(u, v) = t$. That is,

$$\begin{aligned} c(K \cup L) = c(K) + c(L) + t &\leq \frac{3}{2}(\sum_{i \in K \cap R} z_i(i) - t) + \frac{3}{2}(\sum_{i \in L \cap R} z_i(i) - t) + t \\ &\leq \frac{3}{2}(\sum_{i \in (K \cup L) \cap R} z_i(i) - t) \end{aligned} \tag{5}$$

Suppose at time t, a merging of type 2 occurs: the Steiner vertex v hits the simplex. First observe that there are exactly 2 components v is linked to. The algorithm maintains that a Steiner vertex is linked to *at most* 2 components. Moreover, if there is only 1 component v is linked to, the rate of growth of sum of its coordinates is the same as that for the terminals and the Steiner vertex won't hit the simplex.

Call the two components K and L. Let the links to K and L be (v, i) and (v, j) respectively. Without loss of generality let (v, i) be the shorter one with $c(v, i) =: a \leq c(v, j) =: b \leq t$. Note that the increase in the left hand side is $a + b$. That is, $c(K \cup L \cup v) = c(K) + c(L) + (a + b)$. Now note that whenever a link from v is removed (this happens only when two components linked to the same Steiner vertex merge), we always remove the costlier link from v. Thus, the smaller link (v, i) is in fact the *first* link of v. That is, before time a, the coordinates of v were all 0. By the running of the algorithm, a Steiner vertex is linked to at most two components at all times. Thus, the rate of increase of sum of coordinates is at most 2. Therefore at time t when the vertex hits the simplex, the sum of coordinates (which is t since it hits the simplex) is at most $2(t - a)$. This gives a bound on t: $2(t - a) \geq t$ implying $t \geq a/2$. Furthermore, note that $t \geq b$ since the terminal j links to a vertex v at time $c(j, v) = b$ which must be before t. This gives us

$$c(K \cup L \cup v) \leq c(K) + c(L) + \frac{3}{2}t \leq \frac{3}{2}(\sum_{i \in (K \cup L) \cap R} z_i(i) - t)$$

where the last inequality follows as in Inequality 5.

Suppose a merging of type 3 happens at time t: a Steiner vertex links to components K, L, M via links $(v, i), (v, j)$ and (v, l) respectively. Each of these links have cost less than t. Thus the cost of the new component

$$c(K \cup L \cup M \cup v) \leq c(K) + c(L) + c(M) + 3t \leq \frac{3}{2}(\sum_{i \in (K \cup L \cup M)} z_i(i) - t)$$

where the last inequality follows from by induction on the components K, L, M. This completes the proof. □

2.4.1 Implementation in $\tilde{O}(|E|)$ time

We now describe an implementation of the Algorithm SMART-EMBED which runs in almost linear time ($O((|E| + |V|) \log |V|)$. Note that SMART-EMBED has a notion of time, and this implementation mimics the same by only maintaining the times at which the merge operations of SMART-EMBED take place.

For every edge $(u, v) \in E$, we maintain a variable $t(u, v) := c(u, v)$ the cost of the edge. The idea is that the edge (u, v) comes to play in SMART-EMBED precisely at time $t = t(u, v)$. For instance, if the vertices u and v are required, at time $t = t(u, v)$ the components containing them merge. Moreover a Steiner vertex v links to a component at time $t(u, v)$ for some terminal u in the component.

We also maintain a time-variable $t(v)$ for every Steiner vertex v which is not in any component. $t(v)$ stands for the time at which the Steiner vertex v will hit the simplex if no other merge steps occur in between. Initially $t(v)$ is set to ∞ (which for implementation issues can be assumed to be an integer larger than sum of all edge weights).

We maintain a list L of $t(u, v)$'s (for all edges (u, v)) and $t(v)$'s (for Steiner vertices v). These are precisely the set of times at which the merge steps of SMART-EMBED take place. We remark that the list is not static as $t(v)$'s can change as the algorithm progresses. However, we always maintain L sorted in non-decreasing order of times. The implementation reads the list L in the sorted order and depending on the entry read, and whenever applicable, performs one of the three merge steps of SMART-EMBED. We show that these merges can all be made to run with an amortized $O(\log |V|)$ time per entry in L, and since there are at most $O(|E| + |V|)$ entries in L, the implementation runs in almost linear time of $O((|E| + |V|) \log |V|)$.

Data Structures:
We now describe the data structure used for the implementation. The data structures are similar to disjoint set data structures implemented via linked lists (see for example, [28]). These are used in an implementation of Kruskal's minimum spanning tree algorithm.

For each vertex v of the graph, we maintain 4 pointers: *head* - which points to the connected component containing v, *next* - which points to the next vertex in v's connected component, *copy1* and *copy2* - these point to two copies of the same vertex, as we see these will be used by only Steiner vertices and for terminals the two pointers will always be `null`.

The components $K \in \mathcal{K}$ are stored as linked lists of vertices. For every component, one of the vertices will be an identifier vertex. Each vertex in the component will point to this identifier vertex via their *head* pointers. Thus, given a vertex v, the component which

contains it can be found in $O(1)$ time. Initially, we start with $|V|$ linked lists, one for each vertex. However, there are only $|R|$ (number of terminals) components corresponding to the terminals; the *head* pointers of the Steiner vertices are initialized to `null`. For each component, we also maintain the size of the component. We also maintain a pointer *tail* which points to the last vertex in the list.

For each Steiner vertex $v \in S$, we store two copies of it, call them v_A and v_B. These correspond to the two possible components a Steiner vertex could link to. Each of these copies are linked with the original vertex via their *copy* pointers. Whenever a Steiner vertex v is picked in a component, these copies are deleted and the Steiner vertex is treated like a terminal henceforth.

For every component K, we maintain another linked-list, STEINER(K), which contains the list of all Steiner vertices K is linked to. The list contains only copies of the Steiner vertices and not the original Steiner vertex. STEINER(K) is stored similarly as K is stored: a linked list with all Steiner vertices pointing to an identifier vertex for STEINER(K). The lists K and STEINER(K) are doubly linked to each other for all K, thus given a Steiner vertex v one can find the components to which it is linked to in $O(1)$ operations. In fact, we will call this function LINK(v) which returns a set (which we also call LINK(v), abusing notation) of (identifiers of) connected components linked to v. LINK(v) is of at most size two, for all Steiner vertices v. For any Steiner vertex v, we also maintain a set of edges LINK-EDGES(v) which contains the *unique* links of v to the various components in LINK(v).

To calculate $t(v)$, we maintain a bunch of variables for every Steiner vertex v. We store a variable $z(v)$ for every Steiner vertex which is supposed to contain the sum-of-coordinates of v as in SMART-EMBED. $z(v)$ is initialized to 0 for every Steiner vertex. We maintain a variable last(v), which is the last time when a modification had to be made to $z(v)$. These will correspond to instances when v links to a component, or when two components linked to v merge. last(v) is also initialized to 0. We maintain rate(v), which is the rate at which $z(v)$ increases since last(v). *rate*(v) will always equal to $|\text{LINK}(v)|$. Using last(v), rate(v) and $z(v)$, at any time $t(u, v)$, one can easily figure out $t(v)$: if rate(v) $= 1$, $t(v) = \infty$, otherwise $t(v) = t(u, v) - z(v)$.

We now describe how SMART-EMBED can be implemented using these data structures. The algorithm details are in Algorithm 3. Initially, the set of components $\mathcal{K}$ is initialized to be all singleton terminals. That is, all terminals have their *head* pointer pointing to themselves. All $t(v)$'s are set to ∞. The list L is read in non-decreasing order. If the entry read is $t(u, v)$ for some edge (u, v) and u and v are not in the same component, then the following is done. If u and v are both terminals, the components corresponding to them are merged and (u, v) is added to the tree. The set of Steiner vertices STEINER($K(u)$) and STEINER($K(v)$) are also merged and for Steiner vertices v linked to both $K(u)$ and $K(v)$, $z(v)$, last(v), rate(v)

and $t(v)$ are modified. If u is a terminal and v is a vertex not linked to $K(u)$, then either v links to $K(u)$ if $|\textsc{Link}(v)| \leq 1$, or a merge step corresponding to merge step 3 of SMART-EMBED takes place. If the entry of L read corresponds to $t(v)$ for some Steiner vertex v, then it means v has hit the simplex in SMART-EMBED. The components in $\textsc{Link}(v)$ are merged and the link-edges LINK-EDGES(v) are added to the tree. The algorithm terminates when there is a single component formed.

What remains to be described is how to perform the merge steps in $O(\log |V|)$ amortized time per entry of L. It is known that if the sets to be merged are always disjoint, then this can be done using the disjoint-set data structures. In our case, any two components are disjoint. However, for two components K_1 and K_2, the list of linked Steiner vertices $\textsc{Steiner}(K_1)$ and $\textsc{Steiner}(K_2)$ might not be disjoint and the above argument for disjoint data structures fail. Nevertheless, we can show that the total time taken by the merge steps can be bounded by $O(|E| + |V| \log |V|)$. The main idea is this: if the two sets $\textsc{Steiner}(K_1)$ and $\textsc{Steiner}(K_2)$ have a "big" enough union, then the argument for disjoint set data structures can be made to go through. If the union is not "big", then the intersection must be "big". Note that a vertex v in the intersection of $\textsc{Steiner}(K_1)$ and $\textsc{Steiner}(K_2)$ is a vertex linked to both K_1 and K_2. When two such components merge, the costlier of the two link-edges in LINK-EDGES(v) is discarded and this never plays a role in the algorithm subsequently. Thus, all the operations (which are only $O(1)$) made on v in this merge operation can now be charged to the costlier edge in LINK-EDGES(v).

Algorithm 3 Implementation of SMART EMBED

1. **Initialization:**

   - For every vertex $u \in V$, create linked list containing u with pointers *head*, *next*, *copy1*, *copy2*. For terminals $i \in R$, *head* points to itself, *next* is initialized to null, and the two copy pointers are always set to null. We also initialize the *tail* pointers for each to point to itself.
   - For Steiner vertices $v \in S$, *head* and *next* both point to null. For each v, create two linked lists v_A, v_B which are the two copies of v. The two copy pointers of v point to v_A and v_B respectively. All pointers of the copy nodes are set to null.
   - For every Steiner vertex $v \in S$, create linked list LINK(v) containing two pointers initially set to null. Also create a list LINK-EDGES(v) initially set to null. Initialize variables $z(v)$ to 0, last(v) to 0, rate(v) to 0 and $t(v)$ to ∞.
   - Initialize the tree T to be a linked list of edges initialized to an empty list. L be the list of $t(u, v)$'s and $t(v)$'s. Along with the value of the $t()$, we also store the edge or the Steiner vertex responsible for it. L can be stored as a heap so that insertion in sorted can be done in $O(\log |L|)$ time and the list can be read out in a non-decreasing order with $O(1)$ time per read.

2. Read the entries of L in non-decreasing order. If the entry read corresponds to a Steiner vertex (that is $t(v)$ for some Steiner vertex v), then merge the components v is linked to along with the Steiner vertex v: For $K_1, K_2 \in$ LINK(v), **Merge**(K_1, K_2, v). Delete the copies v_A, v_B, delete $t(v)$ from L, and add the links of v to the tree, that is, append LINK-EDGES(v) to T.

3. If the next term in the list L is the edge $(u, v), then$

   (a) If u and v are in the same component, delete $t(u, v)$ from L and proceed to the next entry.

   (b) If $u \in R, v \in R$, merge the two components containing them and add the edge to the tree: **Merge**$(K(u), K(v))$ and append (u, v) to T.

   (c) If $u \in R, v \in S$ (or vice-versa) and $K(u) \notin$ LINK(v), that is, v is not already linked to the component containing u

      i. If $K(v)$ is not null, merge the components containing u and v and add the edge (u, v) to the tree: **Merge**$(K(u), K(v))$ and append (u, v) to T.

      ii. If $|$LINK(v)$| = 2$, that is, v is already linked to 2 other components, merge those with the component containing u and add v to the new component, **Merge**$(K(u),$ LINK(v)$, v)$. Add the edge (u, v) and edges in LINK-EDGES(v) to the tree: append (u, v) and LINK-EDGES(v) to T. Also delete the copies of v and delete $t(v)$ from L, as in Step 2.

      iii. If $|$LINK(v)$| \leq 1$, then do the following: append the edge (u, v) to LINK-EDGES(v); append a copy of v (choose arbitrarily if both copies have their heads pointing to null) to STEINER($K(u)$); append $K(u)$ to LINK(v); recalculate $z(v) = z(v) + \text{rate}(v) \cdot (t(u, v) - \text{last}(v))$, last($v$) $= t(u, v)$ and rate(v) $=$ rate(v) $+ 1$. Recalculate $t(v)$ to ∞ if (new) rate(v) $= 1$, or $t(v) = (t(u, v) - z(v))$ and place it in the correct order in L.

What remains to be described in the implementation is the **Merge**(K_1, K_2) step. When two components merge, a number of things needs to be taken care of: firstly, the two linked lists corresponding to the components need to be merged. Furthermore, the linked lists STEINER(K_1) and STEINER(K_2) need to be merged *and* care needs to be taken that the duplicate copies of the same Steiner vertex should be removed. Moreover, for precisely these Steiner vertices which were linked to both K_1 and K_2, the variables $z(v), \text{last}(v), \text{rate}(v)$ and $t(v)$ need to be recalculated.

The merge step is very similar to the weighted-union step used in disjoint-set data structures. Since our sets (STEINER(K_1) and STEINER(K_2)) might not be disjoint, we need a little more care. However, recognizing the duplicates only takes constant times more time since the recognition requires going through the vertices of one component, and the weighted union step does precisely that.

We now describe merge step in Algorithm 4.

Algorithm 4 Merge(K_1, K_2)

---

1. Choose the smaller linked list (which can be determined the size variable) of K_1 and K_2, say it is K_1. Append K_1 to K_2 using the *tail* pointer of K_2. Update the *tail* pointer of K_2 to point to the last element of K_1. For every vertex in K_1, make the *head* pointer point to the identifier of K_2

2. Choose the smaller linked list among STEINER(K_1) and STEINER(K_2), say STEINER(K_2) and append STEINER(K_2) to STEINER(K_1) as in the merge of K_1 and K_2. Also make the identifier of K_2 (which now starts a list of the merged K_1 and K_2) point to the identifier of STEINER(K_1) (which now starts a list of the merged STEINER(K_1) and STEINER(K_2)), and vice-versa. This takes care that the LINK(v) operation is consistent with the merges.

   For every vertex v (or rather copy of v) in STEINER(K_2) do the following: check if a copy of it exists in STEINER(K_1). This can be done in $O(1)$ pointer chasing: for a vertex v_A in STEINER(K_2), go to the original Steiner vertex v and figure out the component containing the other copy. If a copy does not exist, move the *head* pointer of v (the copy) to the identifier of STEINER(K_1).

   If a copy exists, and the original Steiner vertex be v, then this means that v is linked to both K_1 and K_2. Remove the copy of v (call it v_A) from STEINER(K_2) and make the *head* pointer of v_A point to `null`. Also remove the costlier edge from LINK-EDGES(v). Furthermore, for v recalculate the variables $z(v), \text{last}(v), \text{rate}(v), t(v)$ as in last step of the implementation above. The only difference being $\text{rate}(v)$ decreases by 1 (note that $|\text{LINK}(v)|$ also decreases by 1 too).

Theorem 2.4.2 *Given a weighted quasi-bipartite graph $G(V = R \cup S, E; c)$, Algorithm 3 runs in time $O((|E| + |V|)\log(|V|))$ and returns a tree T with cost within $3/2$ times the optimal Steiner tree.*

Proof: It is easy to see that the implementation mimics Algorithm 2 and thus from Theorem 2.4 the tree T returned is within 3/2 times the optimal tree. We need to argue about the running time.

Note that the sorting of L (storing it as a heap) takes $O((|E| + |V|)\log(|V|))$ time. Suppose that for every entry of L, all the operations of Algorithm 3 took $O(\log |V|)$ time. Then since the size of L is at most $(|E| + |V|)$ we would be done. In fact, it is easy to check that all the steps take $O(\log |V|)$ time except the merge step. We now show that the total time taken in merge steps across the run of the algorithm is $O(|E| + |V| \log |V|)$ and we will be done.

When two components K_1, K_2 are merged, one individual step of merge might take $O(\min(|K_1|, |K_2|) + \min(|\textsc{Steiner}(K_1)|, |\textsc{Steiner}(K_2)|)$ time which is linear. However, since the smaller list is always appended to the larger list, an amortized analysis is possible. Indeed this the most naive implementation of the "union" step in the disjoint step data structures.

Consider the appending of K_1 and K_2. Note that every vertex which changes its *head* pointer ends up in a component at least twice the size of the original component it was in. This is because it was in the smaller component to begin with and the components are disjoint. Thus, since the maximum size of a component in $|V|$, each vertex changes its head pointer at most $O(\log |V|)$ times implying the total number of *head* pointer changes for vertices in components throughout the run of the algorithm is at most $O(|V| \log |V|)$.

However the same accounting does not hold for the appending of $\textsc{Steiner}(K_1)$ and $\textsc{Steiner}(K_2)$. This is because the two sets need not be disjoint and thus the doubling argument does not hold. Nevertheless, we can argue the total number of operations when we append two Steiner lists is also bounded by $O(|E| + |V| \log |V|)$. This is because if the union of the two lists $\textsc{Steiner}(K_1)$ and $\textsc{Steiner}(K_2)$ is not too large (say it is smaller than 1.5 times the size of the smaller list), then the *intersection* will have to be large (larger than 0.5 times the size of the smaller list). Moreover, for every Steiner vertex v in the intersection, we remove one edge from $\textsc{link-edges}(v)$ and this edge never appears again in the algorithm.

Thus, whenever a Steiner vertex changes its *head* pointer, if it moves into a list larger than 1.5 times the list it was in, we charge the movement of the pointer to the vertex. Thus a single Steiner vertex can be charged at most $O(\log |V|)$ times. If the Steiner vertex which changes its *head* pointer doesn't move into a list larger than 1.5 times the list it was in, then we charge the movement of *head* pointers of *all* the Steiner vertices in the smaller list equally to the Steiner vertices in the intersection. Thus by the above observation about intersections being large, every Steiner vertex in the intersection is charged 2. Moreover, vertices in the intersection perform $O(1)$ operations re-evaluating the various variables ($z(v)$, last(v),etc), thus the extra 2 can be swept in the $O(1)$. Now, for every Steiner vertex in the intersection,

we move the $O(1)$ charge on it, on to the *unique* link-edge incident to it that is removed. The crucial observation is that this link-edge is never brought back on to the algorithm, and thus the charge on it remains $O(1)$.

Therefore, adding up, the total number of operations for merge operations can be charged to vertices and edges, each vertex having charge $O(\log |V|)$ and each edge having charge $O(1)$. The theorem follows. □

2.5 A $\sqrt{2}$ Factor Approximation Algorithm

In this section and the next, we give algorithms for the Steiner tree problem using EMBED as a black-box unlike the 3/2-factor algorithm. We will require nothing more than the statement of Theorem 2.3.4. Both algorithms would use EMBED in rounds to obtain *useful* Steiner vertices, although each round will be preceded by a pre-processing step. In this section we look at a $\sqrt{2}$-factor algorithm.

Notation 1 *$MST(U; c)$ denotes the minimum cost spanning tree on vertices U given the costs c. Based on the context, it also denotes the cost of this tree.*

We start by giving a high level idea of our algorithm. The algorithm will finally return a cost c_2 and a subset of Steiner vertices $X \subseteq S$ such that

1. The optimal Steiner tree w.r.t. c_2 is the MST on the terminals. Equivalently, by Theorem 2.3.4, EMBED when run on G, c_2 terminates with a feasible embedding z with $\gamma(z) = MST(R; c_2)$.
2. $MST(X \cup R; c) \leq \sqrt{2} \cdot MST(R; c_2)$

The costs c_2 will be only smaller than c; therefore, z is also feasible for c. Hence, the two conditions imply that we get a factor $\sqrt{2}$ approximation.

Initially, $X = \emptyset$ and we obtain c_2 by reducing the costs of the required-required edges by a factor of $\sqrt{2}$ and leaving the costs of required-Steiner edges unchanged. We denote the reduced cost at this point as c_1 which we use later. Clearly Condition 2 is satisfied now, and will remain an *invariant* of the algorithm.

Suppose that condition 1 is not satisfied, that is, when EMBED is run on G, c_2, a Steiner vertex $v \in S$ hits the simplex. At this point, the algorithm adds v to X, and will modify c_2 by reducing the costs of certain required-required edges further, as detailed below. This has the effect that if EMBED is run with these new costs, v does not hit the simplex, while still maintaining the invariant. Hence in each iteration, a new Steiner vertex is added to X, implying termination in at most $|S|$ rounds.

We now give the intuition behind modifying the costs so that the invariant is maintained. The first step of scaling all the required-required edges acts as a "global filter" which filters

out Steiner vertices that only help a little. If a Steiner vertex v now hits the simplex, then adding it to X reduces the cost of $MST(R \cup X; c)$ so much that decreasing the cost of required-required edges "local" to it to $\frac{1}{2}$ of the original costs still maintains the invariant. This requires an involved argument (Theorem (2.5.1)) that amortizes the improvements due to all the vertices previously added to X. This has the additional required effect that v itself is filtered out.

Now the formal description of the algorithm follows.

Definition 4 *Applying the* global filter *with parameter $\rho > 1$ gives a cost c_1 defined as $c_1(i,j) = \frac{c(i,j)}{\rho}$ for all $i, j \in R$, and $c_1(i,v) = c(i,v)$ for all $i \in R$ and $v \in S$.*

Definition 5 *Applying a* local *filter w.r.t $X \subseteq S$ gives a cost c_2. Let $T_1 = MST(R \cup X; c_1)$, and for each $u \in X$, $Clos(u)$ denote the closest required vertex to u. The cost c_2 after applying the local filter w.r.t X is defined as $c_2(Clos(u), j) = \frac{1}{2}c(Clos(u), j)$ (half the original cost), for every $u \in X$ and $j \in R$ ($j \neq Clos(u)$) that is adjacent to u in T_1. $c_2(e) = c_1(e)$ (the global filter is retained) otherwise.*

Algorithm 5 PRIMAL-DUAL

1. Apply global filter with parameter $\rho = \sqrt{2}$ to get c_1.
   Initialize $X \leftarrow \varnothing$; $c_2 \leftarrow c_1$.
2. **Repeat** till EMBED returns z
   Run EMBED on G, c_2.
   If EMBED returns v then
   $X = X \cup v$; Apply local filter w.r.t X to get c_2.
3. Return $T_1 = MST(R \cup X; c_1), z$.

Theorem 2.5.1 *The algorithm* PRIMAL-DUAL *terminates in at most $|S|$ rounds, returning a Steiner tree T_1 and a feasible embedding z of G, c such that $c(T_1) \leq \sqrt{2} \cdot \gamma(z) \leq \sqrt{2} \cdot OPT$.*

Proof: Let $T_1 = E_0 \cup E_1$, where E_0 denotes the required-Steiner edges and E_1 denotes the required-required edges of T_1. We bound the costs of these two sets separately. Let E_2 be the set of edges modified by the local filter, that is, e such that $c_2(e) = \frac{1}{2}c(e)$. Call such edges *diminished*. Define T_2 to be $E_2 \cup E_1$. It can be shown that T_2 is an $MST(R, c_2)$.

Claim 2.5.2 *T_2 is an MST of R with respect to cost c_2.*

Proof: By definition of the local filter, it is clear that $E_2 \cup E_1$ is a terminal spanning tree. If it is not a minimum one w.r.t cost c_2, there exists an edge $e \in E(T_2)$ and an edge

$f \notin E(T_2)$ such that $T_2 - e + f$ is spanning and $c_2(f) < c_2(e)$. Note that $c_2(f) = c_1(f)$, since $f \notin E_2$. Also, since $f \notin E_1 \cup E_0$, $T_1 + f$ has a cycle, and in fact it is the precisely the cycle formed in $T_2 + f$ with the edges in E_2 replaced by length 2 paths containing a Steiner vertex. Moreover, by definition of the local filter, there will be an edge e' in the cycle in $T_1 + f$ such that $c_1(e') \geq c_2(e)$, and thus $T_1 + f - e'$ will be a cheaper spanning tree of $R \cup X$ w.r.t. costs c_1, contradicting the choice of T_1. □

Thus, from Theorem 2.3.4, $c_2(T_2) = \gamma(z)$. We have $c(T_1) = c(E_0) + c(E_1)$, $c_2(T_2) = c_2(E_2) + c_2(E_1)$ and it is enough to prove that

- $c(E_0) \leq \sqrt{2} c_2(E_2)$.
  This is essentially a consequence of the observation that $c_1(T_1) \leq MST(R; c_1)$. Since $T_2 = E_1 \cup E_2$ is also a spanning tree of R, we get $c_1(T_1) \leq c_1(T_2)$. Expanding the costs, we get
  $$c_1(E_0) + c_1(E_1) \leq c_1(E_2) + c_1(E_1).$$
  Since E_0 are required vertex-Steiner edges, $c_1(E_0) = c(E_0)$. $c_1(E_2) = c(E_2)/\sqrt{2} = \sqrt{2} c_2(E_2)$ by definition, giving us $c(E_0) \leq \sqrt{2} c_2(E_2)$.

- $c(E_1) \leq \sqrt{2} c_2(E_1)$.
  Since E_1 costs are not modified by the local filter, $c_2(E_1) = c_1(E_1)$ and in fact the relation holds with equality.

□

In fact, the above algorithm has a faster implementation. Although the algorithm constructs the set X in a certain order, it turns out that the order does not matter. Hence it is enough to simply apply the global filter and go through the Steiner vertices (in any order) once, picking the ones that help.

Algorithm 6 REDUCED ONE-PASS HEURISTIC

1. Apply global filter with parameter $\rho = \sqrt{2}$ to get c_1.
   Initialize $X \leftarrow \varnothing$;
2. For all $v \in S$,
   If $MST(R \cup X \cup v; c_1) < MST(R \cup X; c_1)$, then
   $X = X \cup v$;
3. Return $T_1 = MST(R \cup X; c_1)$.

Theorem 2.5.3 *There exists a feasible embedding z of G, c such that for T_1 returned by Algorithm* REDUCED ONE-PASS HEURISTIC, $c(T_1) \leq \sqrt{2} \cdot \gamma(z)$.

The proof of Theorem (2.5.3) is similar to Theorem (2.5.1). Note that the above algorithm makes at most $|S|$ minimum spanning tree computations and is hence is very efficient. In particular, it runs in strongly polynomial time.

Proof: Let T_1 be the tree returned by Algorithm REDUCED ONE-PASS HEURISTIC with X as the set of Steiner vertices in T_1. Firstly observe that for any vertex $v \in S \setminus X$, $MST(R \cup X \cup v; c_1) \geq MST(R \cup X; c_1)$. This is because, a Steiner vertex which does not help the MST on $R \cup X$ at an earlier iteration *cannot* help the MST at a later iteration; the heaviest edge in the unique path between any two terminals in T_1 keeps on decreasing. Note that we use the fact here that the graph is quasi-bipartite and a Steiner vertex can connect only to terminals.

As in the proof of Theorem 2.5.1, let $T_1 = E_0 \cup E_1$. Construct the costs c_2 by applying the local filter w.r.t X. Let E_2 be the set of diminished edges and let $T_2 = E_1 \cup E_2$. Once again, one can apply the same proof of Claim 2.5.2 to show that $T_2 = MST(R, c_2)$. Moreover,

Claim 2.5.4 *For every Steiner vertex $v \in S$, $MST(R \cup v; c_2) \geq MST(R; c_2) = c_2(T_2)$.*

Proof: Suppose for some Steiner vertex v, $MST(R \cup v; c_2) < MST(R; c_2)$. Thus there exist a set of edges A, each of the form (v, j) where j is a terminal; and a set of terminal-terminal edges in $E(T_2)$ so that $c_2(A) = c_1(A) < c_2(B)$; and $T_2 \setminus B \cup A$ is a tree spanning $R \cup v$. Now, among the edges in B, some are diminished and some are in $E(T_1)$; and each diminished edge e corresponds to two edges e_1 and e_2 in $E(T_1)$ one of which, say e_1, has $c_1(e_1) \geq c_2(e)$. Thus from B, we get a subset of edges $B' \subseteq E(T_1)$ with $c_1(B') \geq c_2(B)$. Moreover, note that $T_1 \setminus B' \cup A$ is a valid spanning tree of $R \cup X \cup v$. This implies $MST(R \cup X \cup v; c_1) < MST(R \cup X; c_1)$, which is a contradiction. □

Thus, in the quasi-bipartite graph G with costs c_2, T_2 is an MST spanning the terminals and the addition of no Steiner vertex improves it. So, by Theorem 2.3.1, we know that running EMBED on $(G; c_2)$ will return a feasible dual z with $c_2(T_2) = \gamma(z)$. Since c_2 is only reduced, this embedding is a feasible embedding of (G, c) as well. The proof ends by noting that $c(T_1) \leq \sqrt{2} c_2(T_2)$ which is exactly as in the proof of Theorem 2.5.1. □

2.6 A $\frac{4}{3}$ Factor Approximation Algorithm

The primal-dual $\frac{4}{3}$ approximation algorithm is along the lines of the one in the previous section, with the major difference being that it drops Steiner vertices from X when beneficial. The other differences are that it applies the global filter with $\rho = 4/3$, and the definition of a local filter is somewhat different. And like the earlier algorithm, the order of vertices picked/dropped does not matter. As a result it can be implemented as a local search algorithm with an extra global filtering step, which is what we present here.

Algorithm 7 REDUCED-LOCAL-SEARCH

1. Apply global filter with parameter $\rho = 4/3$ to get c_1.
   Initialize $X \leftarrow \varnothing$, $T_1 = MST(R; c_1)$;

2. Repeat
   If $\exists v$ such that $MST(R \cup X \cup v; c_1) < c_1(T_1)$, $X = X \cup v$.
   If $\exists v$ such that $MST(R \cup X \setminus v; c_1) < c_1(T_1)$, $X = X \setminus v$.
   $T_1 = MST(R \cup X; c_1)$.
   Until No such v exists.

3. Return T_1.

The plain local search algorithm (without the global filtering step) was studied [84] who showed that this algorithm gives a 3/2 factor approximation for quasi-bipartite graphs. This factor is tight. So the simple modification of applying a global filter provably improves the performance of this algorithm. It was shown in [85] that this algorithm can be implemented efficiently.

We show that T_1 returned by the algorithm is within 4/3 of the optimal by exhibiting an embedding z of value greater than 3/4 times the cost of T_1. As in Section (2.5), the analysis proceeds by defining cost c_2 and constructing tree T_2. The factor 4/3 comes from the parameter ρ used in the global filter and the following property of T_1.

Lemma 2.6.1 *The degree of every Steiner vertex in T_1 is at least 4.*

Proof: It is easy to see that T_1 doesn't have vertices of degree 1 or 2. Suppose there existed a Steiner vertex $v \in T_1$ with $deg(v) = 3$. Let a, b, c be the required vertices connected to v and assume $c_1(va) \leq c_1(vb) \leq c_1(vc)$ without loss of generality. Now by triangle inequality property of c, we know $c(va) + c(vb) \geq c(ab)$. Since $c(va) = c_1(va)$ and $c(vb) = c_1(vb)$, we get $\frac{3}{4}(c_1(va) + c_1(vb)) \geq \frac{3}{4}c(ab) = c_1(ab)$. Similarly $\frac{3}{4}(c_1(va) + c_1(vc)) \geq c_1(ac)$. Thus $c_1(ab)+c_1(ac) \leq \frac{3}{4}(2c_1(va)+c_1(vb)+c_1(vc)) \leq c_1(va)+c_1(vb)+c_1(vc)$. Thus $MST(R \cup X)$ would choose (ab) and (ac), rather than choosing $(va), (vb), (vc)$. □

Theorem 2.6.2 *For the tree T_1 returned by* REDUCED-LOCAL-SEARCH*, there exists a feasible embedding z such that $c(T_1) \leq \frac{4}{3} \cdot \gamma(z)$.*

Proof:

As in the proof of Theorem (2.5.1), denote the edges of T_1 as $E_1 \cup E_0$. Define c_2 as: For every Steiner vertex $v \in T_1$ and for every $j \neq Clos(v)$ connected to v in T_1, let $c_2(Clos(v), j) = c_1(vj)$. Note that $c_1(vj) \leq c_1(Clos(v), j)$, for otherwise T_1 would have picked $(Clos(v), j)$ instead of (vj). Call these required vertex-required vertex edges

diminished. For every other edge, $c_2(e) := c_1(e)$. Let E_2 be the set of diminished edges and let $T_2 := E_1 \cup E_2$, be a required vertex spanning tree. By the conditions of the algorithm, since T_1 is an MST of $R \cup X$ with costs c_1 and no Steiner vertices help X, T_2 is an MST of R with costs c_2 and no Steiner vertex helps T_2. Thus, by Theorem (2.3.1) running EMBED on G, c_2 returns a feasible embedding z of value $c_2(T_2)$. We now bound the cost of T_1.

We have $c(T_1) = c(E_1) + c(E_0) = c(E_1) + c_1(E_0)$. Note that $c_2(T_2) = c_1(E_1) + c_2(E_2)$ since E_1 is not diminished. As in the proof of Theorem (2.5.1), we argue term by term. By definition we have $c(E_1) = \frac{3}{4}c_1(E_1)$.

Every Steiner vertex $v \in T_1$ contributes $deg(v) - 1$ edges to E_2 and $deg(v)$ edges in E_0, where $deg(v)$ is the degree of v in T_1. By definition the $deg(v)-1$ edges have cost exactly the cost of the largest $deg(v)-1$ edges of the $deg(v)$ edges it contributes to E_0. By lemma (2.6.1), $deg(v) \geq 4$ and thus we get $c_1(E_0) \leq \frac{3}{4}c_2(E_2)$. Adding, we get $c(T_1) \leq \frac{4}{3}c_2(T_2) = \frac{4}{3}\gamma(z)$. □

2.7 Discussion

In this chapter, we gave a new way of lower bounding the cost of the minimum Steiner tree in a graph via the simplex embedding LP (LP SimpEmb). We showed that for any graph, this lower bound equals the lower bound obtained by the bidirected cut relaxation; in fact we saw that the simplex embedding LP was in a sense equivalent to the dual of the bidirected cut relaxation. We used this geometric way of looking at the dual to give faster and better upper bounds on the integrality gap for a class of graphs called quasi-bipartite graphs.

Clearly the most important question to address is whether the geometric approach to the bidirected cut relaxation described here can be extended to general graphs. In fact, there is a natural generalization of the EMBED procedure described above to the case where there are Steiner-Steiner edges; however, it has not yielded any results for the general case. As noted above, one crucial property possessed by quasi-bipartite graphs is Theorem (2.3.1): if the minimum spanning tree on the terminals is the optimal Steiner tree, then the relaxation is exact. However, this property is not satisfied by general graphs. An example is given in Figure 2.2.2: the MST on the terminals is optimal but the bidirected cut relaxation has a gap of 16/15. An interesting question would be upper bounding the gap in such instances, and then perhaps our techniques of reducing costs may be useful.

CHAPTER III

APPROXIMABILITY OF MAXIMUM BUDGETED ALLOCATIONS

3.1 Introduction

Resource allocation problems of distributing a fixed supply of resources to multiple agents in an "optimal" manner are ubiquitous in computer science and economics. In this chapter we consider the following *maximum budgeted allocation* (MBA) problem: Given a set of m indivisible items and n agents; each agent i willing to pay b_{ij} on item j and with a maximum budget of B_i, the goal is to allocate items to agents to maximize revenue.

The problem naturally arises as a revenue maximization problem for the auctioneer in an auction with budgeted agents. Examples of such auctions (see, for example [9]) include those used for the privatization of public assets in western Europe, or those for the distribution of radio spectra in the US, where the magnitude of the transactions involved put financial or liquidity constraints on bidders. With the growth of the Internet, budget-constrained auctions have gained increasing relevance. Firstly, e-auctions held on the web (on e-Bay, for instance) cater to the long-tail of users who are inherently budget-constrained. Secondly, sponsored search auctions hosted by search engines (Google, Yahoo!, MSN and the like), where advertisers bid on keywords, include budget specification as a feature. A common (and natural) assumption in keyword auctions that is typically made is that bids of advertisers are much smaller than the budgets. However, with the extension of the sponsored search medium from the web onto the more classical media, such as radio and television[1] where this assumption is not as reasonable, the general budget-constrained auctions need to be addressed.

MBA is known to be NP-hard — even in the case of two bidders it is not hard to see that MBA encodes PARTITION [2]. In this chapter we study the approximability of MBA and improve upon the best known approximation and hardness of approximation factors. Moreover, we use our hardness reductions to get better hardness results for other allocation problems like submodular welfare maximization(SWM), generalized assignment problem (GAP) and maximum spanning star-forest (MSSF).

3.1.1 Maximum Budgeted Allocation

We start with the formal problem definition.

[1] see for instance `http://www.google.com/adwords/audioads/` and `http://www.google.com/adwords/tvads`

[2] PARTITION: Given n integers $a_1, \cdots, a_n$ and a target B, decide whether there is a subset of these integers adding up to exactly B

Definition 2 *Let Q and A be a set of m indivisible items and n agents respectively, with agent i willing to pay b_{ij} for item j. Each agent i has a budget constraint B_i and on receiving a set $S \subseteq Q$ of items, pays* $\min(B_i, \sum_{j \in S} b_{ij})$. *An* allocation $\Gamma : A \to 2^Q$ *is the partitioning the sets of items Q into disjoint sets $\Gamma(1), \cdots, \Gamma(n)$. The* maximum budgeted allocation *problem, or simply MBA, is to find the allocation which maximizes the total revenue, that is,* $\sum_{i \in A} \min(B_i, \sum_{j \in \Gamma(i)} b_{ij})$.

Note that we can assume without loss of generality that $b_{ij} \leq B_i, \ \forall i \in A, j \in Q$. This is because if bids are larger than budget, decreasing it to the budget does not change the value of any allocation. Sometimes, motivated by the application, one can add the constraint that $b_{ij} \leq \beta \cdot B_i$ for all $i \in A$ and $j \in Q$, for some $\beta \leq 1$. We call such an instance β-MBA.

Previous and Related Work: As noted above, MBA is NP-hard and this observation was made concurrently by many authors ([41, 87, 4, 70]). The first approximation algorithm for the problem was given by Garg, Kumar and Pandit[41] who gave a $2/(1+\sqrt{5})(\simeq 0.618)$ factor approximation. Andelman and Mansour[4] improved the factor to $(1-1/e)(\simeq 0.632)$. For the special case when budgets of all bidders were equal, [4] improved the factor to 0.717. We refer to the thesis of Andelman[3] for an exposition. Very recently, and independent of our work, Azar et.al. [7] obtained a 2/3-factor for the general MBA problem. They also considered a *uniform* version of the problem where for every item j, the bid of any agent is either b_j (independent of the agent) or 0. They gave a $1/\sqrt{2}(\simeq 0.707)$ factor for the same. All these algorithms are based on a natural LP relaxation (LP(6) in Section 3.1.3) which we use as well.

In the setting of sponsored search auctions, MBA, or rather β-MBA with $\beta \to 0$, has been studied mainly in an online context. Mehta et.al.[77], and later Buchbinder et.al.[13], gave $(1 - 1/e)$-competitive algorithms when the assumption of bids being small to budget is made. The dependence of the factor on β is not quite clear from either of the works. Moreover, as per our knowledge, nothing better was known concerning the approximability of the *offline* β-MBA than what was suggested by algorithms for MBA.

Our results: We give two approximation algorithms for MBA. The first, based on iterative LP rounding, attains a factor of 3/4. The algorithm described in Section 3.2. The second algorithm, based on the primal-dual schema, is faster and attains a factor of $3/4(1 - \epsilon)$, for any $\epsilon > 0$. The running time of the algorithm is $\tilde{O}(\frac{nm}{\epsilon})^3$, and is thus almost linear for constant ϵ and dense instances. We describe the algorithm in Section 3.3. Our algorithms can be extended suitably for β-MBA as well giving a $1-\beta/4$ factor approximation algorithm.

[3] the ˜ hides logarithmic factors

In Section 3.4, we show it is NP hard to approximate MBA to a factor better than 15/16 via a gap-preserving reduction from MAX-3-LIN(2). Our hardness instances are *uniform* in the sense of Azar et.al. [7] implying uniform MBA is as hard. Our hardness reductions extend to give a $(1-\beta/16)$ hardness for β-MBA as well. Interestingly, our reductions can be used to obtain better inapproximability results for other problems: SWM (15/16 hardness even with demand queries), GAP (10/11 hardness) and MSSF(10/11 and 13/14 for the edge and node weighted versions), which we elaborate below.

3.1.2 Relations to other allocation problems

Submodular Welfare Maximization (SWM): As in the definition of MBA, let Q be a set of m indivisible items and A be a set of n agents. For agent i, let $u_i : 2^Q \to \mathbb{R}_+$ be a utility function where for a subset of items $S \subseteq Q$, $u_i(S)$ denote the utility obtained by agent i when S is allocated to it. Given an allocation of items to agents, the total *social welfare* is the sum of utilities of the agents. The *welfare maximization* problem is to find an allocation of maximum social welfare.

Before discussing the complexity of the welfare maximization problem, one needs to be careful of how the utility functions are represented. Since it takes exponential (in the number of items) size to represent a general set-function, *oracle access* to these functions are assumed and the complexity of the welfare maximization problem depends on the strength of the oracle. The strongest such oracle that has been studied is the so-called *demand oracle*: for any agent i and prices $p_1, p_2, \cdots, p_m$ for all items in Q, returns a subset $S \subseteq Q$ which maximizes $(u_i(S) - \sum_{j \in S} p_j)$.

Welfare maximization problems have been extensively studied (see, for example, [10]) in the past few years with various assumptions made on the utility functions. One important set of utility functions are *monotone submodular utility functions*. A utility function u_i is submodular if for any two subsets S, T of items, $u_i(S \cup T) + u_i(S \cap T) \leq u_i(S) + u_i(T)$. The welfare maximization problem when all the utility functions are submodular is called the submodular welfare maximization problem or simply SWM. Feige and Vondrák [35] gave an $(1 - 1/e + \rho)$-approximation for SWM with $\rho \sim 0.0001$ and showed that it is NP-hard to approximate SWM to better than 275/276.[4]

MBA is a special case of SWM. This follows from the observation that the utility function $u_i(S) = \min(B_i, \sum_{j \in S} b_{ij})$ when B_i, b_{ij}'s are fixed is a submodular function. In Section 3.4.1, we show that in the hardness instances of MBA, the demand oracle can be simulated in poly-time and therefore the 15/16 hardness of approximation for MBA implies a 15/16-hardness of approximation for SWM as well.

[4]We remark that SWM with a different oracle, the *value oracle* which given a set and an agent returns the utility of the agent for the set, has recently been resolved. There was a $(1-1/e)$ hardness given by Khot et.al.[60] and recently Vondrák[94] gave a matching polynomial time algorithm.

Generalized Assignment Problem (GAP): GAP is a problem quite related to MBA: Every item j, along with the bid (profit) b_{ij} for agent (bin) i, also has an inherent size s_{ij}. Instead of a budget constraint, each agent (bin) has a capacity constraint C_i which defines feasible sets: A set S is feasible for (bin) i if $\sum_{j \in S} s_{ij} \leq C_i$. The goal is to find a revenue (profit) maximizing feasible assignment. The main difference between GAP and MBA is that in GAP we are *not allowed* to violate capacity constraints, while in MBA the budget constraint only caps the revenue. As was noted by Chekuri and Khanna[21], a 1/2 approximation algorithm was implicit in the work of Shmoys and Tardos[89]. The factor was improved by Fleischer et.al.[36] to $1 - 1/e$. In the same paper [35] where they give the best known algorithm for SWM, Feige and Vondrák[35] also give a $(1 - 1/e + \rho')$ algorithm for GAP ($\rho' \leq 10^{-5}$). The best known hardness for GAP was $1 - \epsilon$, for some small ϵ which was given by Chekuri and Khanna [21] via a reduction from maximum 3D-matching. Improved hardness results for maximum 3D matching by Chlebik and Chlebikova[24], imply a 422/423 hardness for GAP.

Although MBA and GAP are in some sense incomparable problems, we can use our hardness techniques to get a 10/11 factor hardness of approximation for GAP in Section 3.4.3.

Maximum Spanning Star-Forest Problem (MSSF): Given an undirected unweighted graph G, the MSSF problem is to find a forest with as many edges such that each tree in the forest is a star - all but at most one vertex of the tree are leaves. The edge-weighted MSSF is the natural generalization with weights on edges. The node-weighted MSSF has weights on vertices and the weight of a star is the weight on the leaves. If the star is just an edge, then the weight of the star is the maximum of the weights of the end points.

The unweighted and edge-weighted MSSF was introduced by Nguyen et.al [79] who gave a 3/5 and 1/2-approximation respectively for the problems. They also showed APX hardness of the unweighted version. Chen et.al. [22] improved the factor of unweighted MSSF to 0.71 and introduced node-weighted MSSF giving a 0.64 factor algorithm for it. They also give a 31/32 and 19/20 hardness for the node-weighted and edge-weighted MSSF problems.

Although, at the face of it, MSSF does not seem to have a relation with MBA, once again our hardness technique can be used to improve the hardness of node-weighted and edge-weighted MSSF to 13/14 and 10/11, respectively. We describe this in Section 3.4.4.

3.1.3 The LP Relaxation for MBA

One way to formulate MBA as an integer program is the following:

$$\max\{\sum_{i\in A}\pi_i :\ \pi_i = \min(B_i, \sum_{j\in Q} b_{ij}x_{ij}),\ \forall i;\quad \sum_{i\in A} x_{ij} \le 1, \forall j;\quad x_{ij}\in\{0,1\}\ \}$$

Relaxing the integrality constraints to non-negativity constraints gives an LP relaxation for the problem. We work with the following equivalent LP relaxation of the problem. The equivalence follows by noting that in there exists an optimal fractional solution, $B_i \ge \sum_{j\in Q} b_{ij}x_{ij}$. This was noted by Andelman and Mansour[4] and a similar relaxation was used by Garg et.al. [41].

$$\max\{\sum_{i\in A, j\in Q} b_{ij}x_{ij} :\quad \forall i\in A,\ \sum_{j\in Q} b_{ij}x_{ij} \le B_i;\quad \forall j\in Q,\ \sum_{i\in A} x_{ij}\le 1;\quad \forall i\in A, j\in Q,\ x_{ij}\ge 0\} \tag{6}$$

We remark that the assumption $b_{ij} \le B_i$ is crucial for this LP to be of any use. Without this assumption it is easy to construct examples having arbitrarily high integrality gaps. Consider the instance with one item, n agents each having a budget 1 but bidding n on the item. The LP has a solution of value n while the maximum welfare is obviously 1.

Moreover, the integrality gap of this LP is at most 3/4. In the following example in Figure(3.1.3), the maximum revenue obtained by any feasible allocation is 3 while the value of the LP is 4. The example is due to [4] and thus our main result shows that the integrality gap is exactly 3/4.

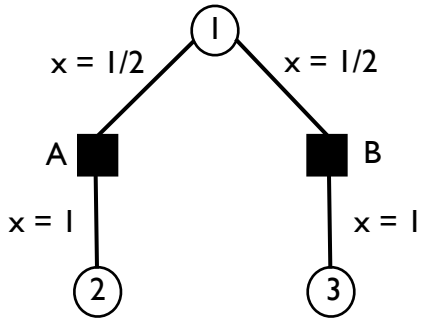

Figure 5: Integrality gap example for LP (6)

Agents are black squares and have budget 2. The bids of agent A and B on item 1 is 2. A bids 1 on 2 and B bids 1 on 3. Note the agent who doesn't get item 1 will spend only 1 and thus the maximum allocation is 3. The LP however gets 4 as shown by the solution x.

3.2 An iterative rounding algorithm for MBA

Let $\mathcal{P}$ be a problem instance defined by the bids and budgets of every agent, that is $\mathcal{P} := (\{b_{ij}\}_{i,j}, \{B_i\}_i)$. With $\mathcal{P}$, we associate a bipartite graph $G(\mathcal{P}) = (A \cup Q, E)$, with $(i,j) \in E$ if i bids on j.

Let $x^*(\mathcal{P})$ be an extreme point solution to LP(6) for the problem instance $\mathcal{P}$. For brevity, we omit writing the dependence on $\mathcal{P}$ when the instance is clear from context. Let E^* be the support of the solution, that is $E^* := \{(i,j) \in E : x^*_{ij} > 0\}$. Note that these are the only *important* edges - one can discard all bids of agents i on item j when $(i,j) \notin E^*$. This does not change the LP optimum and a feasible integral solution in this instance is a feasible solution of the original instance. Call the set of neighbors of an agent i in $G[E^*]$ as $\Gamma(i)$. Call an agent *tight* if $\sum_j b_{ij} x^*_{ij} = B_i$.

The starting point of the algorithm is the following claim about the structure of the extreme point solution. Such an argument using polyhedral combinatorics, was first used in the machine scheduling paper of Lenstra, Shmoys and Tardos [71]. A similar claim can be found in the thesis of Andelman [3].

Claim 3.2.1 *The graph, $G[E^*]$, induced by E^* can be assumed to be a forest. Moreover, except for at most one, all the leaves of a connected component are items. Also at most one agent in a connected component can be* non-tight*.*

Proof: Consider the graph $G[E^*]$. Without loss of generality assume that it is a single connected component. Otherwise we can treat every connected component as a separate instance and argue on each of them separately. Thus, $G[E^*]$ has $(n+m)$ nodes. Also since there are $(n+m)$ constraints in the LP which are not non-negativity constraints, therefore support of any extreme point solution can be of size atmost $(n+m)$. This follows from simple polyhedral combinatorics: at an extreme point, the number of inequalities going tight is at least the number of variables. Since there are only $(n+m)$ constraints which are not non-negativity constraints, all but at most $(n+m)$ variables must satisfy the non-negativity constraints with equality, that is, should be 0. Thus $|E^*| \leq n+m$.

Hence there is at most one cycle in $G[E^*]$. Suppose the cycle is: $(i_1, j_1, i_2, j_2, \cdots, j_k, i_1)$, where $\{i_1, \cdots, i_k\}$ and $\{j_1, \cdots, j_k\}$ are the subsets of agents and items respectively. Consider the feasible fractional solution obtained by decreasing x^* on (i_1, j_1) by ϵ_1 and increasing on (i_2, j_1) by ϵ_1, decreasing on (i_2, j_1) by ϵ_2 , increasing on (i_2, j_1) by ϵ_2, and so on. Note that if the ϵ_i's are small enough, the item constraints are satisfied. The relation between ϵ_1 and ϵ_2 (and cascading to other ϵ_r's) is: $\epsilon_1 b_{i_2,j_1} = \epsilon_2 b_{i_2,j_2}$, that is, the fraction of money spent by i_2 on j_1 equals the money freed by j_2. The exception is the last ϵ_k, which might not satisfy the condition with ϵ_1. If $\epsilon_k b_{i_1,j_k} > \epsilon_1 b_{i_1,j_1}$, then just stop the increase on the edge (i_1, j_k) to the point where there is equality. If $\epsilon_k b_{i_1,j_k} < \epsilon_1 b_{i_1,j_1}$, then start the whole procedure by increasing x^* on the edge (i_1, j_1) instead of decreasing and so on. In one of

the two cases, we will get a feasible solution of equal value and the ϵ_i's can be so scaled so as to reduce x^* on one edge to 0. In other words, the cycle is broken without decreasing the LP value.

Thus, $G[E^*]$ is a tree. Moreover, since $(n+m-1)$ edges are positive, there must be $(n+m-1)$ equalities among the budget and the item constraints. Thus at most one budget constraint can be violated which implies at most one agent can *non tight*. Now since the bids are less than the budget, therefore if an agent is a leaf of the tree $G[E^*]$ then he must be *non-tight*. Hence atmost one agent can be a leaf of the tree $G[E^*]$. □

Call an item a *leaf item*, if it is a leaf in $G[E^*]$. Also call an agent i a *leaf agent* if, except for at most one, all of his neighboring items in $E^*(\mathcal{P})$ are leaves. Note the above claim implies each connected component has at least one leaf item and one leaf agent: in any tree there are two leaves both of which cannot be agents, and there must be an agent with all but one of its neighbors leaves and thus leaf items. For the sake of understanding, we first discuss the following natural iterative algorithm which assigns the leaf items to their neighbors and then adjusts the budget and bids to get a new *residual problem*.

1/2-approx algorithm: Solve $LP(\mathcal{P})$ to get $x^*(\mathcal{P})$. Now pick a leaf agent i. Assign all the leaf items in $\Gamma(i)$ to i. Let j be the unique non-leaf item (if any) in $\Gamma(i)$. Form the new instance $\mathcal{P}'$ by removing $\Gamma(i) \setminus j$ and all incident edges from $\mathcal{P}'$. Let $b = \sum_{l \in \Gamma(i) \setminus j} b_{il}$, be the portion of budget spent by i. Now modify the budget of i and his bid on j in $\mathcal{P}'$ as follows: $B'_i := B_i - b$, and $b'_{ij} := \min(b_{ij}, B'_i)$. Its instructive to note the drop $(b_{ij} - b'_{ij})$ is at most b. (We use here the assumption bids are always smaller than budgets). Iterate on the instance $\mathcal{P}'$.

The above algorithm is a 1/2-approximation algorithm. In every iteration, we show that the revenue generated by the items allocated is at least 1/2 of the drop in the LP value $(LP(\mathcal{P}) - LP(\mathcal{P}'))$. Suppose in some iteration, i be the leaf agent chosen, and let j be the its non-leaf neighbor, and let the revenue generated by algorithm be b. Note that x^*, the solution to $\mathcal{P}$, restricted to the edges in $\mathcal{P}'$ is still a feasible solution. Thus the drop in the LP is: $b + (b_{ij} - b'_{ij})x_{ij}$. Since $(b_{ij} - b'_{ij})$ is atmost b, and x_{ij} at most 1, we get $LP(\mathcal{P}) - LP(\mathcal{P}') \leq 2b$.

To prove a better factor in the analysis, one way is to give a better bound on the drop, $(LP(\mathcal{P}) - LP(\mathcal{P}'))$. Unfortunately, the above analysis is almost tight and there exists an example (Figure 6 below) where the LP drop in the first iteration is $\simeq$ twice the revenue generated by the algorithm in that iteration.

Thus, for an improved analysis for this algorithm, one needs a better amortized analysis across different iterations rather than analyzing iteration-by-iteration. This seems non-trivial as we solve the LP again at each iteration and the solutions could be very different across iterations making it harder to analyze over iterations.

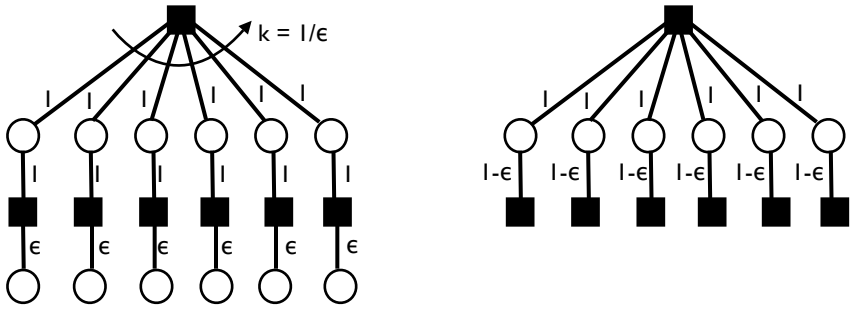

Figure 6: Example showing LP drop almost twice as value obtained by naive algorithm.

For some $\epsilon > 0$ let $k = 1/\epsilon$. In the instance, there are $k+1$ agents with budgets 1 denoted by black squares and $2k$ items with bids as shown on the figure in the left. The LP value of this is $k+1$: the ϵ edges have $x = 1$, the 1 edges forming a matching have $x = 1-\epsilon$ and the rest have $x = \epsilon$. After the first iteration, the leaf items are assigned and the value obtained is $k\epsilon = 1$. The budgets of the k agents at the bottom reduce to $1-\epsilon$ and so do their modified bids, as shown on the figure in the right. The LP solution for this instance is $k-1+\epsilon$ ($k-1$ items going to bottom $k-1$ agents and the remaining item to the top guy). The LP drop is $2-\epsilon$ and thus is twice the value obtained as $\epsilon \to 0$.

Instead, we modify the above algorithm by defining the *residual problem* $\mathcal{P}'$ in an non-trivial manner. After assigning leaf items to agent i, we do not decrease the budget by the amount assigned, but keep it a little "larger". Thus these agents *lie* about their budgets in the subsequent rounds, and we call these *lying* agents. Since the budget doesn't drop too much, the LP value of the residual problem doesn't drop much either. A possible trouble would arise when items are being assigned to lying agents since they do not pay as much as they have bid. This leads to a trade-off and we show by suitably modifying the residual problem one can get a 3/4 approximation. We now elaborate.

Given a problem instance $\mathcal{P}_0 := \mathcal{P}$, the algorithm proceeds in stages producing newer instances at each stage. On going from $\mathcal{P}_i$ to $\mathcal{P}_{i+1}$, at least one item is allocated to some agent. Items are never de-allocated, thus the process ends in at most m stages. The *value* of an item is defined to be the payment made by the agent who gets it. That is, $value(j) = \min(b_{ij}, B_i - spent(i))$, where $spent(i)$ is the value of items allocated to i at the time j was being allocated. We will always ensure the condition that a lying agent i bids on at most one item j. We will call j the *false item* of i and the bid of i on j to be i's *false bid*. In the beginning no agent is lying.

We now describe the k-th iteration of the iterative algorithm which we call MBA-Iter (Algorithm 8).

Claim 3.2.2 *In each step, at least one item is allocated and thus* MBA-Iter *terminates*

Algorithm 8 k-th step of MBA-ITER

1. Solve $LP(\mathcal{P}_k)$. Remove all edges which are not in $E^*(\mathcal{P}_k)$. These edges will stay removed in all subsequent steps.

2. If there is a lying agent i with $x^*_{ij} = 1$ for his false item j, assign item j to him. In the next instance, $\mathcal{P}_{k+1}$, remove i and j. Proceed to $(k+1)$-th iteration.

3. If there is a non-lying agent i such that all the items in $\Gamma(i)$ are leaf items. Then allocate $\Gamma(i)$ to i. Remove i, $\Gamma(i)$ and all the incident edges to get the new instance $\mathcal{P}_{k+1}$ and proceed to $(k+1)$-th iteration step.

4. Pick a *tight* leaf agent i. Notice that i must have at least two items in $\Gamma(i)$, otherwise tightness would imply that the unique item is a leaf item and thus either step 2 or step 3 must have been performed. Moreover, exactly one item in $\Gamma(i)$ is not a leaf item, and let j be this unique non-leaf item. Allocate all the items in $\Gamma(i) \setminus j$ to i. In $\mathcal{P}_{k+1}$, remove $\Gamma(i) \setminus j$ and all incident edges. Also, modify the budget and bids of agent i. Note that agent i now bids *only* on item j as there are no other edges incident to i. Let the new bid of agent i on item j be

$$b'_{ij} := \max(0, \frac{4b_{ij}x^*_{ij} - B_i}{3x^*_{ij}})$$

   Let the new budget of agent i be $B'_i := b'_{ij}$. Call i lying and j be his false item. Proceed to $(k+1)$-th iteration.

in m steps.

Proof: We show that one of the three steps 2,3 or 4 is always performed and thus some item is always allocated. Consider any component. If a component has only one agent i, then all the items in $\Gamma(i)$ are leaf items. If $\Gamma(i)$ has more than two items, then the agent cannot be lying since the lying agent bids on only one item and Step 3 can be performed. If $\Gamma(i) = \{j\}$, then $x^*_{ij} = 1$ since otherwise x^*_{ij} could be increased giving a better solution. Thus Step 2 or 3 can always be performed depending on if i is lying or not. If the component has at least two agents, then it must have two leaf agents. This can be seen by rooting the tree at any item. At least one of them, say i, is tight by Claim 3.2.1. Thus Step 4 can be performed. □

Theorem 3.2.3 *Given a problem instance $\mathcal{P}$, the allocation obtained by algorithm* MBA-ITER *attains value at least $\frac{3}{4} \cdot LP(\mathcal{P})$.*

Proof: Let $\Delta_k := LP(\mathcal{P}_k) - LP(\mathcal{P}_{k+1})$ denote the drop in the optimum across the k-th iteration. Denote the set of items allocated at step k as Q_k. Note that the total value of the algorithm is $\sum_{j \in Q} value(j) = \sum_k (\sum_{j \in Q_k} value(j))$. Also, the LP optimum of the original

solution is $LP(\mathcal{P}) = \sum_k \Delta_k$ since after the last item is allocated the LP value becomes 0. The following lemma proves the theorem. □

Lemma 3.2.4 *In every stage k, $value(Q_k) := \sum_{j \in Q_k} value(j) \geq \frac{3}{4}\Delta_k$.*

Proof: Items are assigned in either Step 2,3 or 4. Let us analyze Step 2 first. Let i be the lying agent obtaining his false item j. Since $x^*_{ij} = 1$ and lying agents bid on only one item, the remaining solution (keeping the same x^* on all remaining edges) is a valid solution for the LP in $\mathcal{P}_{k+1}$. Thus

$$LP(\mathcal{P}_k) - LP(\mathcal{P}_{k+1}) \leq b',$$

where b' is the false bid of lying agent i on item j. Let b be the bid of agent i on item j, before it was made lying. Then, from Step 4 we know that $b' := \frac{4bx-B}{3x}$, where x was the fraction of item j assigned to i and B is the budget of i. Moreover, the portion of budget spent by i is at most $(B - bx)$. This implies $value(j) \geq bx$. The claim follows by noting for all $b \leq B$ and all x,[5]

$$bx \geq \frac{3}{4} \cdot \frac{4bx - B}{3x}$$

In Step 3, in fact the LP drop equals the value obtained - both the LP drop and the value obtained is the sum of bids on items in $\Gamma(i)$ or B_i, whichever is less.

Coming to step 4, $Q_k = \Gamma(i) \setminus j$ be the set of goods assigned to the tight, non-lying leaf agent i. Let b and b' denote the bids of i on j before and after the step: b_{ij} and b'_{ij}. Let x be x^*_{ij}. Note that $x^*_{il} \leq 1$ for all $l \in Q_k$. Also, x^* restricted to the remaining goods still is a feasible solution in the modified instance $\mathcal{P}_{k+1}$. Since the bid on item j changes from b to b', the drop in the optimum is at most

$$LP(\mathcal{P}_k) - LP(\mathcal{P}_{k+1}) \leq (\sum_{l \in Q_k} b_{il}) + (bx - b'x)$$

Note that $value(Q_k) = \sum_{l \in Q_k} b_{il} \geq B - bx$ by tightness of i. We now show $(bx - b'x) \leq \frac{1}{3} \cdot value(Q_k)$ which would prove the lemma.

If $b' = 0$, this means $4bx \leq B$. Thus, $value(Q_k) \geq B - bx \geq 3bx$. Otherwise, we have

$$(bx - b'x) = bx - \frac{4bx - B}{3x} \cdot x = \frac{B - bx}{3} \leq value(Q_k)/3$$

implying the claim, as before. □

3.3 Primal-dual algorithm for MBA

In this section we give a faster primal-dual algorithm for MBA although we lose a bit on the factor. The main theorem of this section is the following:

[5] $4bx^2 - 4bx + B = b(2x-1)^2 + (B-b) \geq 0$

Theorem 3.3.1 *For any $\epsilon > 0$, there exists an algorithm which runs in $\tilde{O}(nm/\epsilon)$ time and gives a $\frac{3}{4} \cdot (1-\epsilon)$-factor approximation algorithm for MBA.*

Let us start by taking the dual of the LP relaxation LP(6).

$$DUAL := \min\{\sum_{i \in A} B_i \alpha_i + \sum_{j \in Q} p_j : \quad \forall i \in A, j \in Q;\ p_j \geq b_{ij}(1-\alpha_i); \quad \forall i \in A, j \in Q;\ p_j, \alpha_i \geq 0\} \tag{7}$$

We make the following interpretation of the dual variables: Every agent retains α_i of his budget, and all his bids are modified to $b_{ij}(1-\alpha_i)$. The price p_j of a good is the highest modified bid on it. The dual program finds retention factors to minimize the sum of budgets retained and prices of items. We start with a few definitions.

Definition 3 *Let $\Gamma : A \to 2^Q$ be an allocation of items to agents and let the set $\Gamma(i)$ be called the items* owned *by i. Let $S_i := \sum_{j \in \Gamma(i)} b_{ij}$ denote the total bids of i on items in $\Gamma(i)$. Note that the revenue generated by Γ from agent i is $\min(S_i, B_i)$. Given α_i's, the* prices generated by Γ *is defined as follows: $p_j = b_{ij}(1-\alpha_i)$, where j is owned by i. Call an item* wrongly allocated *if $p_j < b_{lj}(1-\alpha_l)$ for some agent l, call it* rightly allocated *otherwise. An allocation Γ is called* valid *(w.r.t α_i's) if all items are rightly allocated, that is, according to the interpretation of the dual given above, all items go to agents with the highest modified bid ($b_{ij}(1-\alpha_i)$) on it. Note that if Γ is valid, (p_j,α_i)'s form a valid dual solution. Given an $\epsilon > 0$, Γ is* ϵ-valid *if $p_j/(1-\epsilon)$ satisfies the dual feasibility constraints with the α_i's.*

Observe that given α_i's; and given an allocation Γ and thus the prices p_j generated by it, the objective of the dual program can be treated agent-by-agent as follows

$$DUAL = \sum_i Dual(i), \quad \text{where } Dual(i) = B_i\alpha_i + \sum_{j \in \Gamma(i)} p_j = B_i\alpha_i + S_i(1-\alpha_i) \tag{8}$$

Now we are ready to describe the main idea of the primal-dual schema. The algorithm starts with all α_i's set to 0 and an allocation valid w.r.t to these. We will "pay-off" this dual by the value obtained from the allocation agent-by-agent. That is, we want to pay-off $Dual(i)$ with $\min(B_i, S_i)$ for all agents i. Call an agent *paid for* if $\min(B_i, S_i) \geq \frac{3}{4} Dual(i)$. We will be done if we find α_i's and an allocation valid w.r.t these such that all agents are paid for.

Let us look at when an agent is paid for. From the definition of $Dual(i)$, an easy calculation shows that an agent is paid for iff $S_i \in [L(\alpha_i), U(\alpha_i)] \cdot B_i$, where $L(\alpha) = \frac{3\alpha}{1+3\alpha}$ and $U(\alpha) = \frac{4-3\alpha}{3-3\alpha}$. Note that S_i depends on Γ which was chosen to be valid w.r.t. α_i's. Moreover, observe that increasing α_i can only lead to the decrease of S_i and vice-versa. This suggests the following next step: for agents i which are unpaid for, if $S_i > U(\alpha_i)B_i$, increase α_i and if $S_i < L(\alpha_i)B_i$, decrease α_i and modify Γ to be the valid allocation w.r.t the α_i's.

However, it is hard to analyze the termination of an algorithm which both increases and decreases α_i's. This is where we use the following observation about the function $L()$ and $U()$. (In fact 3/4 is the largest factor for which the corresponding $L()$ and $U()$ have the following property; see Remark 3.3.4 below).

Property 3.3.2 *For all α, $U(\alpha) \geq L(\alpha) + 1$.* [6]

The above property shows that an agent with $S_i > U(\alpha_i)B_i$ on losing a *single item* j will still have $S_i > U(\alpha_i)B_i - b_{ij} \geq (U(\alpha_i) - 1)B_i \geq L(\alpha_i)B_i$, for any $\alpha_i \in [0, 1]$. Also observe that in the beginning when α_i's are 0, $S_i \geq L(\alpha_i)B_i$. Thus if we can make sure that the size of $\Gamma(i)$ decreases by at most one, when the α_i's of an unpaid agent i is increased, then the case $S_i < L(\alpha_i)B_i$ never occurs and therefore we will never have to decrease α's and termination will be guaranteed.

However, an increase in α_i can lead to movement of more than one item from the current allocation of agent i to the new valid allocation. Thus to ensure *steady* progress is made throughout, we move to ϵ-valid allocations and get a $\frac{3}{4} \cdot (1 - \epsilon)$ algorithm.

We now give details of the Algorithm 9.

Algorithm 9 MBA-PD: Primal Dual Algorithm for MBA

Define $\epsilon_i := \epsilon \cdot \frac{1-\alpha_i}{\alpha_i}$. Throughout, p_j will be the price generated by Γ and current α_i's.

1. Initialize $\alpha_i = 0$ for all agents. Let Γ be the allocation assigning item j to agent i which maximizes b_{ij}.

2. Repeat the following till all agents are paid for:

   Pick an agent i who is not paid for (that is $S_i > U(\alpha_i)B_i$), arbitrarily. Repeat till i becomes paid for:

   If i has no wrongly allocated items in $\Gamma(i)$, then increase $\alpha_i \rightarrow \alpha_i(1 + \epsilon_i)$. (Note that when $\alpha_i = 0$, ϵ_i is undefined. In that case, modify $\alpha_i = \epsilon$ from 0.)

   Else pick any one wrongly allocated item j of agent i, and modify Γ by allocating j to the agent l who maximizes $b_{lj}(1 - \alpha_l)$. (Note that this makes j rightly allocated but can potentially make agent l not paid for).

Claim 3.3.3 *Throughout the algorithm, $S_i \geq L(\alpha_i)B_i$.*

Proof: The claim is true to start with ($L(0) = 0$). Moreover, S_i of an agent i decreases *only if* i is not paid for, that is, $S_i > U(\alpha_i)B_i$. Now, since items are transferred one at a

[6] $U(\alpha) - 1 = \frac{1}{3-3\alpha} \geq \frac{3\alpha}{1+3\alpha} =: L(\alpha) \Leftarrow 1 + 3\alpha \geq 9\alpha(1 - \alpha) \Leftarrow 9\alpha^2 - 6\alpha + 1 \geq 0$

time and each item can contribute at most B_i to S_i, the fact $U(\alpha) \geq 1 + L(\alpha)$ for all α proves the claim. □

Remark 3.3.4 *In general, one can compare $Dual(i)$ and $\min(S_i, B_i)$ to figure out what L, U should be to get a ρ-approximation. As it turns out, the largest ρ for which U, L satisfies property 3.3.2 is $3/4$ (and it cannot be any larger due to the integrality gap example). However, the bottleneck above is the fact that each item can contribute at most B_i to S_i. Note that in the case of β-MBA this is $\beta \cdot B_i$ and indeed this is what gives a better factor algorithm. Details in Section 3.3.1.*

Theorem 3.3.5 *For any $\epsilon > 0$, given α_i's, an allocation Γ ϵ-valid w.r.t it and p_j, the prices generated by Γ; if all agents are paid for then Γ is a $3/4(1-\epsilon)$-factor approximation for MBA.*

Proof: Consider the dual solution (p_j, α_i). Since all agents are paid for, $\min(B_i, S_i) \geq 3/4 \cdot Dual(i)$. Thus the total value obtained from Γ is at least $3/4 \sum_{i \in A} Dual(i)$. Moreover, since Γ is ϵ-valid, $(p_j/(1-\epsilon), \alpha_i)$ forms a valid dual of cost $\frac{1}{1-\epsilon} \sum_{i \in A} Dual(i)$ which is an upper bound on the optimum of the LP and thus the proof follows. □

Along with Theorem 3.3.5, the following theorem about the running time proves Theorem 3.3.1.

Theorem 3.3.6 *Algorithm* MBA-PD *terminates in $(nm \cdot \ln(3m)/\epsilon)$ iterations with an allocation Γ with all agents paid for. Moreover, the allocation is ϵ-valid w.r.t the final α_i's.*

Proof: Let us first show the allocation throughout remains ϵ-valid w.r.t. the α_i's. Note that initially the allocation is valid. Subsequently, the price of an item j generated by Γ decreases only when the α_i of an agent i owning j increases. This happens only in Step 2, and moreover j must be rightly allocated before the increase. Now the following calculation shows that after the increase of α_i, p_j decreases by a factor of $(1-\epsilon)$. Thus, $(p_j/(1-\epsilon), \alpha_i)$'s form a valid dual solution implying Γ is ϵ-valid.

$$\begin{aligned} p_j^{(new)} &= b_{ij}(1 - \alpha_i^{(new)}) = b_{ij}(1 - \alpha_i(1+\epsilon_i)) \\ &= b_{ij}(1-\alpha_i)(1 - \epsilon_i\alpha_i/(1-\alpha_i)) = p_j^{(old)}(1-\epsilon) \end{aligned}$$

Now in Step 2, note that until there are agents not paid for, either we decrease the number of wrongly allocated items or we increase the α_i for some agent i. That is, in at most m iterations of Step 2, α_i of some agent becomes $\alpha_i(1+\epsilon_i)$. Now, note that if $\alpha_i > 1 - 1/3m$ for some agent, he is paid for. This follows simply by noting that $S_i \leq mB_i = U(1-1/3m) \cdot B_i$ and the fact that $S_i \geq L(\alpha_i)B_i$, for all α_i.

Claim 3.3.7 *If α_i is increased $t > 0$ times, then it becomes $1 - (1-\epsilon)^t$.*

Proof: At $t = 1$, the claim is true as α_i becomes ϵ. Suppose the claim is true for some $t \geq 1$. On the $t+1$th increase, α_i goes to

$$\begin{aligned}\alpha_i(1+\epsilon_i) &= \alpha_i + \epsilon(1-\alpha_i) = \alpha_i(1-\epsilon)+\epsilon \\ &= (1-(1-\epsilon)^t)(1-\epsilon)+\epsilon \\ &= 1-(1-\epsilon)^{t+1}\end{aligned}$$

□

Thus if α_i is increased $\ln(3m)/\epsilon$ times, i becomes paid for throughout the remainder of the algorithm. Since there are n agents, and in each m-steps some agent's α_i increases, in $(nm \cdot \ln(3m)/\epsilon)$ iterations all agents are paid for and the algorithm terminates. □

3.3.1 Extension to β-MBA

The algorithm for β-MBA is exactly the same as Algorithm 9. The only difference is the definition of *paid for* and $L(), U()$. Call an agent paid for if $\min(B_i, S_i) \geq \frac{4-\beta}{4} Dual(i)$. Define the function $L(\alpha) := \frac{\alpha(4-\beta)}{\alpha(4-\beta)+\beta}$ and $U(\alpha) := \frac{(1-\alpha)(4-\beta)+\beta}{(1-\alpha)(4-\beta)}$ Note that when $\beta = 1$, the definitions coincide with the definitions in the previous section.

Claim 3.3.8 *Given α_i's, agent is paid for if $S_i \in [L(\alpha_i), U(\alpha_i)] \cdot B_i$*

Proof: Agent i is paid for if both $B_i \geq \frac{(4-\beta)}{4}(B_i\alpha_i + S_i(1-\alpha_i))$ and $S_i \geq \frac{(4-\beta)}{4}(B_i\alpha_i + S_i(1-\alpha_i))$. Let us lose the subscript for the remainder of the proof.
The first implies

$$S(1-\alpha) \leq B(\frac{4}{(4-\beta)} - \alpha) \Rightarrow S(1-\alpha) \leq B\frac{(4-\beta)(1-\alpha)+\beta}{(4-\beta)} \Rightarrow S \leq U(\alpha)B$$

The second implies

$$S(\frac{4}{(4-\beta)} - (1-\alpha)) \geq B\alpha \Rightarrow S\frac{\alpha(4-\beta)+\beta}{(4-\beta)} \geq B\alpha \Rightarrow S \geq L(\alpha)B$$

□

Property 3.3.9 *For all α, $U(\alpha) \geq L(\alpha) + \beta$*

Proof: Note that $U(\alpha) = 1 + \frac{\beta}{(1-\alpha)(4-\beta)}$ and $L(\alpha) = 1 - \frac{\beta}{\alpha(4-\beta)+\beta}$. Now,

$$\begin{aligned}U(\alpha) - \beta \geq \quad L(\alpha) &\Leftarrow \frac{\beta}{(1-\alpha)(4-\beta)} - \beta \geq \frac{\beta}{\alpha(4-\beta)+\beta} \\ &\Leftarrow \frac{1}{(1-\alpha)(4-\beta)} \geq \frac{\alpha(4-\beta)-(1-\beta)}{\alpha(4-\beta)+\beta} \\ &\Leftarrow \alpha(4-\beta)+\beta \geq (4-\beta)^2\alpha(1-\alpha) - (1-\alpha)(1-\beta)(4-\beta) \\ \Leftarrow \quad \alpha^2(4-\beta)^2 - \alpha(4-\beta)((1-\beta)+(4-\beta)-1) + \beta + (1-\beta)(4-\beta) &\geq 0 \\ &\Leftarrow (\alpha(4-\beta))^2 - 2\alpha(4-\beta)(2-\beta) + (2-\beta)^2 \geq 0 \\ &\Leftarrow (\alpha(4-\beta) - (2-\beta))^2 \geq 0\end{aligned}$$

which is true for any α. $\square$

Theorem 3.3.10 *The algorithm 9 with the above definitions gives a $(1-\beta/4)(1-\epsilon)$-factor approximation for β-MBA in $\tilde{O}(nm/\epsilon)$ time.*

Proof: Armed with the Property 3.3.9 which implies $S_i \geq L(\alpha)B_i$ for all i, the proof of Theorem 3.3.6 can be modified (the only difference is we need to run till $\alpha_i > 1-1/(4-\beta)m$ instead of $(1-1/3m)$), to show that the algorithm terminates with an ϵ-valid allocation with all agents paid for. The proof of the factor follows from the proof of Theorem 3.3.5 and Claim 3.3.8. $\square$

3.4 Inapproximability of MBA and related problems

In this section we study the inapproximability of MBA and the related problems as stated in the introduction. The main theorem of this section is the following 15/16 hardness of approximation factor for MBA.

Theorem 3.4.1 *For any $\epsilon > 0$, it is NP-hard to approximate* MBA *to a factor $15/16+\epsilon$. This holds even for* uniform *instances.*

We give a reduction from Max-3-Lin(2) to MBA to prove the above theorem. The Max-3-Lin(2) problem is as follows: Given a set of m equations in n variables over $GF(2)$, where each equation contains exactly 3 variables, find an assignment to the variables to maximize the number of satisfied equations. Håstad, in his seminal work [47], gave the following theorem.

Theorem 3.4.2 *[47] Given an instance I of* Max-3-Lin(2)*, for any $\delta, \eta > 0$, its NP hard to distinguish between the two cases:* Yes: *There is an assignment satisfying $(1-\delta)$-fraction of equations, and* No: *No assignment satisfies more than $(1/2+\eta)$-fraction of equations.*

We now describe the main idea of the hardness reduction, the same idea will also be used in the reduction for other problems. For every variable x in an Max-3-Lin(2) instance, we will have two agents corresponding to the variable being 0 or 1. For each such pair of agents we have a *switch item*, an item bid on only by this pair of agents, and the allocation of the item will coincide with the assignment of the variable. For every equation e in the Max-3-Lin(2) instance, we will have items coinciding with the satisfying assignments of the equation. For instance if the equation $e : x+y+z = 0$, we will have have items corresponding to $\langle x:0, y:0, z:0\rangle$, $\langle x:0, y:1, z:1\rangle$ and so on. Each such item will be desired by the three corresponding agents: for example $\langle x:0, y:0, z:0\rangle$ will be wanted by the 0 agent corresponding to x, y and z. The bids and budgets are so set so that the switch items are always allocated and thus each allocation corresponds to an assignment. In this way,

an allocation instance encodes an assignment instance. The hardness of MBA and other allocation problems follows from the hardness of MAX-3-LIN(2). We give the details now.

Let I be an instance of MAX-3-LIN(2). Denote the variables as $x_1, \cdots, x_n$. Also let $deg(x_i)$ be the *degree* of variable x_i i.e. the number of equations in which variable x_i occurs. Note that $\sum_i deg(x_i) = 3m$. We construct an instance $R(I)$ of MBA as follows:

- For every variable x_i, we have two agents which we label as $\langle x_i : 0\rangle$ and $\langle x_i : 1\rangle$, corresponding to the two assignments. The budget of both these agents is $4deg(x_i)$ (4 per equation).
- There are two kinds of items. For every variable x_i, we have a *switch item* s_i. Both agents, $\langle x_i : 0\rangle$ and $\langle x_i : 1\rangle$, bid their budget $4deg(x_i)$ on s_i. No one else bids on s_i.
- For every equation $e : x_i + x_j + x_k = \alpha$ ($\alpha \in \{0, 1\}$), we have 4 kinds of items corresponding to the four assignments to x_i, x_j, x_k which satisfy the equation: $\langle x_i : \alpha, x_j : \alpha, x_k : \alpha\rangle$, $\langle x_i : \alpha, x_j : \bar{\alpha}, x_k : \bar{\alpha}\rangle$, $\langle x_i : \bar{\alpha}, x_j : \bar{\alpha}, x_k : \alpha\rangle$ and $\langle x_i : \bar{\alpha}, x_j : \alpha, x_k : \bar{\alpha}\rangle$. For each equation, we have 3 copies of each of the four items. The set of all 12 items are called *equation items*, and denoted by S_e. Thus we have $12m$ equation items, in all.

  For every equation item of the form $\langle x_i : \alpha_i, x_j : \alpha_j, x_k : \alpha_k\rangle$, only three agents bid on it: the agents $\langle x_i : \alpha_i\rangle$, $\langle x_j : \alpha_j\rangle$ and $\langle x_k : \alpha_k\rangle$. The bids are of value 1 each.

Figure 7 illustrates the reduction above locally on three variables x_1, x_2, x_3 for the equation $x_1 + x_2 + x_3 = 1$.

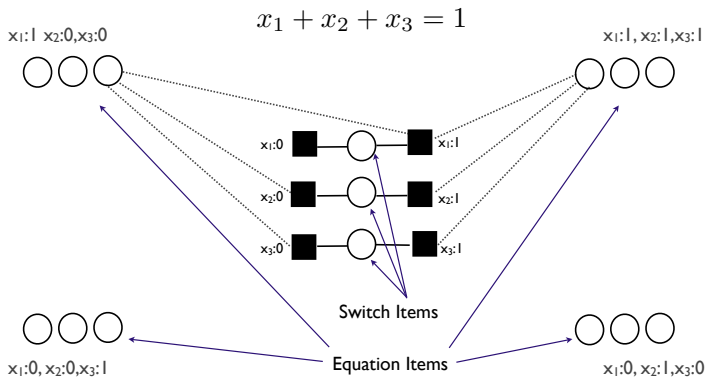

Figure 7: Hardness gadget.
The hardness gadget for reduction of MBA to MAX-3-LIN(2). Dotted lines are a bid of 1 and the solid lines are a bid equalling the budget, $4deg(x_i)$.

We call a solution to $R(I)$ a *valid* assignment if it allocates all the switch items. The following lemma is not hard to see.

Lemma 3.4.3 *There always exists an optimal solution to $R(I)$ in which every switch item is allocated, that is the solution is valid.*

Proof: Suppose there is a solution which is not valid. Thus there is a switch item s_i which is not allocated. Allocating s_i to either $\langle x_i : 0\rangle$ or $\langle x_i : 1\rangle$ and de-allocating the items allocated to the agent can only increase the value of the allocation. □

Suppose $R(I)$ allocates switch item s_i to agent $\langle x_i : 0\rangle$, then we say that $R(I)$ assigns x_i to 1, and similarly if s_i is allocated to $\langle x_i : 1\rangle$ then we say x_i is assigned to 0. Thus by lemma 3.4.3, every optimal solution of $R(I)$ also gives an assignment of variables for I, and we call this the assignment by $R(I)$. Now observe the following property which is used to prove a crucial lemma 3.4.5:

Property 3.4.4 *If $(x_i = \alpha_i, x_j = \alpha_j, x_k = \alpha_k)$ is a satisfying assignment for the equation $x_i + x_j + x_k = \alpha$, then the other three satisfying assignments are $(x_i = \bar{\alpha_i}, x_j = \bar{\alpha_j}, x_k = \alpha_k)$, $(x_i = \bar{\alpha_i}, x_j = \alpha_j, x_k = \bar{\alpha_k})$, and $(x_i = \alpha_i, x_j = \bar{\alpha_j}, x_k = \bar{\alpha_k})$.*

Since agents who get switch items exhaust their budget, any more equation items given to them generate no extra revenue. We say that an equation item can be allocated in $R(I)$ only if it generates revenue, that is, it is not allocated to an agent who has spent all his budget.

Lemma 3.4.5 *Given an assignment of variables by $R(I)$, if an equation e is satisfied then all the 12 items of S_e can be allocated in $R(I)$. Otherwise, at most 9 items of S_e can be allocated in $R(I)$.*

Proof: If an equation e is satisfied, then there must be one equation item $\langle x_i : \alpha_i, x_j : \alpha_j, x_k : \alpha_k\rangle$ such that x_r is assigned α_r $(r = i, j, k)$ in the assignment by $R(I)$ (that is the switch item s_r is given to $\langle x_r : \bar{\alpha_r}\rangle$). Assign the 12 items of S_e as follows: give one the three copies of $\langle x_i : \alpha_i, x_j : \alpha_j, x_k : \alpha_k\rangle$ to agents $\langle x_i : \alpha_i\rangle$, $\langle x_j : \alpha_j\rangle$ and $\langle x_k : \alpha_k\rangle$. Note that none of them have got the switch item. Moreover, for the other items in S_e, give all 3 copies of $\langle x_i : \alpha_i, x_j : \bar{\alpha_j}, x_k : \bar{\alpha_k}\rangle$ to agent $\langle x_i : \alpha_i\rangle$, and similarly for the three copies of $\langle x_i : \bar{\alpha_i}, x_j : \alpha_j, x_k : \bar{\alpha_k}\rangle$ and $\langle x_i : \bar{\alpha_i}, x_j : \bar{\alpha_j}, x_k : \alpha_k\rangle$. Since each agent gets 4 items, he does not exhaust his budget.

If an equation e is *not* satisfied, then observe that there must be an equation item $\langle x_i : \alpha_i, x_j : \alpha_j, x_k : \alpha_k\rangle$ such that x_r is assigned $\bar{\alpha_r}$ $(r = i, j, k)$ in the assignment. That is, all the three agents bidding on this item have their budgets filled up via switch items. Thus none of the copies of this equation item can be allocated, implying at most 9 items can be allocated. □

The following two lemma along with Håstad's theorem prove the hardness for maximum budgeted allocation given in Theorem 3.4.1.

Lemma 3.4.6 *If $OPT(I) \geq m(1-\epsilon)$, then the maximum budgeted allocation revenue of $R(I)$ is at least $24m - 12m\epsilon$.*

Proof: Allocate the switch elements in $R(I)$ so that the *assignment* of variables by $R(I)$ is same as the *assignment* of I. That is, if x_i is assigned 1 in the solution to I, allocate s_i to $\langle x_i : 0 \rangle$, and vice versa if x_i is assigned 0. For every equation which is satisfied, allocate the 12 equation items as described in Lemma(3.4.5). Since each agent gets at most 4 items per equation, it gets at most $4deg(x_i)$ revenue which is under his budget. Thus the total budgeted allocation gives revenue: gain from switch items + gain from equation items $= \sum_i 4deg(x_i) + 12m(1-\epsilon) = 24m - 12m\epsilon$. □

Lemma 3.4.7 *If $OPT(I) \leq m(1/2+\eta)$, then the maximum budgeted allocation revenue of $R(I)$ is at most $22.5m + 3m\eta$*

Proof: Suppose not. i.e . the maximum revenue of $R(I)$ is strictly greater than $22.5m + 3m\eta$. Since the switch items can attain at most $12m$ revenue, $10.5m + 3m\eta$ must have been obtained from equation items. We claim that there must be strictly more than $m(1/2 + \eta)$ equations so that at least 10 out of their 12 equation items are allocated. Otherwise the revenue generated will be at most $12m(1/2 + \eta) + 9m(1/2 - \eta) = 10.5m + 3m\eta$ The contradiction follows from Lemma(3.4.5). □

3.4.1 Hardness of SMW with demand oracle

As noted in Section 3.1.2, MBA is a special case of SMW. Thus the hardness of approximation in Theorem 3.4.1 would imply a hardness of approximation for SMW with the demand oracle, if the demand oracle could be simulated in poly-time in the hard instances of MBA. Lemma 3.4.9 below shows that this indeed is the case which gives the following theorem.

Theorem 3.4.8 *For any $\epsilon > 0$, it is NP-hard to approximate submodular welfare with demand queries to a factor $15/16 + \epsilon$.*

Lemma 3.4.9 *Given any instance I of* Max-3-Lin(2)*, in the corresponding instance $R(I)$ as defined in Section 3.4 the demand oracle can be simulated in polynomial time.*

Proof: We need to show that for any agent i and given prices $p_1, p_2 \cdots$ to the various items, one can find a subset of items S which maximizes $(\min(B_i, \sum_{j \in S} b_{ij}) - \sum_{j \in S} p_j)$. Call such a bundle the optimal bundle. Observe that in the instance $R(I)$, the bid of an agent i is 1 on an equation item and B_i on the switch item. Therefore, the optimal bundle S either consists of just the switch item or consists of B_i equation items. The best equation items are obviously those of the smallest price and thus can be found easily (in particular in polynomial time). □

3.4.2 Hardness of β-MBA

The hardness reduction given above can be easily modified to give a hardness result for β-MBA, for any constant $1 \geq \beta > 0$. Note that the budget of an agent is four times the degree of the analogous variable. We increase the budget of agents $\langle x_i : 0 \rangle$ and $\langle x_i : 1 \rangle$ to $\frac{1}{\beta} 4deg(x_i)$. For each agent, introduce dummy items so that the total bid of an agent on these dummy items is $(\frac{1}{\beta} - 1)$ times its original budget. The rest of the reduction remains the same. Call this new instance β-$R(I)$.

Claim 3.4.10 *We can assume that in any optimal allocation, all the dummy items are assigned*

Proof: If a dummy item is not assigned and assigning it exceeds the budget of the agent implies the agent must be allocated an equation item. De-allocating the equation item and allocating the dummy item gives an allocation of at least the original cost. □

Once the dummy items are assigned, the instance reduces to the original instance. We have the following analogous lemmas of Lemma3.4.6 and Lemma3.4.7.

Lemma 3.4.11 *If $OPT(I) \geq m(1-\epsilon)$, then the maximum budgeted allocation revenue of β-$R(I)$ is at least $24m - 12m\epsilon + 24m(\frac{1}{\beta} - 1)$. If $OPT(I) \leq m(1/2 + \eta)$, then the maximum budgeted allocation revenue of $R(I)$ is at most $22.5m + 3m\eta + 24m(\frac{1}{\beta} - 1)$*

Proof: The extra $24m(\frac{1}{\beta} - 1)$ is just the total value of the dummy items which is obtained in both cases. □

The above theorem with Håstad's theorem gives the following hardness result for β-MBA.

Theorem 3.4.12 *For any $\epsilon > 0$, it is NP-hard to approximate β-MBA to a factor $1 - \beta/16 + \epsilon$.*

3.4.3 Hardness of GAP

To remind, in the generalized assignment problem (GAP) we have n bins each with a capacity B_i. There are a set of items with item j having a profit p_{ij} and size s_{ij} corresponding to bin i. The objective is to find an allocation of items to bins so that no capacities are violated and the total profit obtained is maximized.

One of the bottlenecks for getting a better lower bound for MBA is the extra contribution of switch items which are always allocated irrespective of I. A way of decreasing the effect of these switch items is to decrease their value. In the case of MBA this implies reducing the bids of agents on switch items. Note that this might lead to an agent having a switch item and an equation item as he has budget remaining, and thus the allocation does not correspond to an assignment for the variables. This is where the generality of GAP helps

us: the switch item will have a reduced profit but the size will still be the capacity of the agent (bin). However, since we would want switch items to be *always* allocated, we cannot reduce their profits by too much. We use this idea to get the following theorem.

Theorem 3.4.13 *For any $\epsilon > 0$, it is NP-hard to approximate GAP to a factor $10/11 + \epsilon$.*

We now describe our gadget more in detail. The gadget is very much like the one used for MBA.

- For every variable x_i, we have two bins $\langle x_i : 0\rangle$ and $\langle x_i : 1\rangle$, corresponding to the two assignments. The capacity of both these bins is $2deg(x_i)$ (2 per equation).
- There are two kinds of items. For every variable x_i, we have a *switch item* s_i. s_i can go to only one of the two bins, $\langle x_i : 0\rangle$ and $\langle x_i : 1\rangle$. Its capacity for both bins is $2deg(x_i)$ while its profit is $deg(x_i)/2$.
- For every equation of the form $e : x_i + x_j + x_k = \alpha$ ($\alpha \in \{0, 1\}$), we have a set S_e of 4 items, called *equation items*, corresponding to the four assignments to x_i, x_j, x_k which satisfy the equation: $\langle x_i : \alpha, x_j : \alpha, x_k : \alpha\rangle$, $\langle x_i : \alpha, x_j : \bar{\alpha}, x_k : \bar{\alpha}\rangle$, $\langle x_i : \bar{\alpha}, x_j : \bar{\alpha}, x_k : \alpha\rangle$ and $\langle x_i : \bar{\alpha}, x_j : \alpha, x_k : \bar{\alpha}\rangle$. Thus we have $4m$ equation items, in all. Every equation item of the form $\langle x_i : \alpha_i, x_j : \alpha_j, x_k : \alpha_k\rangle$, can go to any one of the three bins $\langle x_i : \alpha_i\rangle$, $\langle x_j : \alpha_j\rangle$ and $\langle x_k : \alpha_k\rangle$. The profit and size for each of this bins is 1.

We will use $R(I)$ to refer to the instance of GAP obtained from the instance I of Max-3-Lin(2). Lets say a solution to an instance $R(I)$ of GAP is a *k-assignment* solution if exactly k switch items have been allocated. We will use *valid-assignment* to refer to the *n-assignment* solution.

Lemma 3.4.14 *For every solution of $R(I)$ which is a k-assignment solution such that variable x_i is unassigned (i.e. item s_i is neither allocated to $\langle x_i : 0\rangle$ nor $\langle x_i : 1\rangle$) there exists a k-assignment solution of at least the same value in which x_i is unassigned and both $\langle x_i : 0\rangle$ and $\langle x_i : 1\rangle$ together gets atmost one item from S_e for every equation e of x_i. i.e. they both get a total of atmost $deg(x_i)$ items.*

Proof: Suppose not. i.e. there exists an equation e (say $x_i + x_j + x_k = \alpha$) s.t. both $\langle x_i : 0\rangle$ and $\langle x_i : 1\rangle$ together gets atleast two items out of S_e. Notice that the switch items of x_j and x_k can fill the capacity of atmost one bin out of their respective two bins. Suppose the free bins are $\langle x_j : \alpha\rangle$ and $\langle x_k : \alpha\rangle$ (The other cases can be considered similarly). Now except for the item $\langle x_i : \alpha, x_j : \bar{\alpha}, x_k : \bar{\alpha}\rangle$, all the other 3 items in S_e are wanted by $\langle x_j : \alpha\rangle$ and $\langle x_k : \alpha\rangle$. By the above property, all of these 3 items can be allcocated to the bins $\langle x_j : \alpha\rangle$ and $\langle x_k : \alpha\rangle$. Thus we can reallocate the items of S_e such that atmost one item out of S_e is allocated to the corresponding bins of variable x_i without decreasing the profit. □

Now by the above lemma, for any unassigned variable x_i in a *k-assignment* solution, one of the bins out of $\langle x_i : 0\rangle$ and $\langle x_i : 1\rangle$ will have atmost $deg(x_i)/2$ items. We can remove these items and allocate the switch element of x_i without reducing the profit. Thus we get the following corollary.

Corollary 3.4.15 *For every optimal solution of $R(I)$ which is a k-assignment solution there exists a (k+1)-assignment solution which is also optimal. Therefore, there exists a optimal solution of $R(I)$ which is a valid assignment*

Now using arguments similar to lemma 3.4.5 , one can show the following:

Lemma 3.4.16 *Consider a valid-assignment solution (say v-sol) of R(I). If an equation e is satisfied by the assignment of variables given by v-sol then all the 4 items of S_e can be allocated in v-sol. Otherwise atmost 3 items out of S_e can be allocated in v-sol.*

Now using arguments similar to lemma 3.4.6 and 3.4.7, one can prove the following lemma which along with Håstad's theorem implies Theorem 3.4.13.

Lemma 3.4.17 *Let I be an instance of* Max-3-Lin(2) *and R(I) be its reduction to GAP, then:*

- *If $OPT(I) \geq m(1-\epsilon)$, then the maximum profit of $R(I)$ is at least $5.5m - 4m\epsilon$.*
- *If $OPT(I) \leq m(1/2+\eta)$, then the maximum profit of $R(I)$ is at most $5m + m\eta$*

Proof: Suppose $OPT(I) \geq m(1-\epsilon)$. Allocate switch elements in $R(I)$ so that the assignment of variables is same as the one given by optimal solution of I. Now the profit from switch items equals: $\sum_i (deg(x_i)/2) = 3m/2$. Also by lemma 3.4.16, the profit from *equation* items is atleast $4m(1-\epsilon)$. Combining both, we get the first part of the lemma.

Suppose $OPT(I) \leq m(1/2+\eta)$. By corollary 3.4.15, there exists a optimum solution of $R(I)$ which is a *valid* assignment. Consider any such solution. Now the claim is that for no more than $m(1/2+\eta)$ equations, all the 4 items of S_e's can be allocated in this solution. If they do, then along with lemma 3.4.16 it contradicts the fact that $OPT(I) \leq m(1/2+\eta)$. Thus profit from equation items can be atmost: $4m(1/2+\eta) + 3m(1/2-\eta) = 7m/2 + m\eta$. Hence total profit can be atmost $3m/2 + 7m/2 + m\eta$. □

3.4.4 Hardness of weighted MSSF

Given an undirected graph G, the unweighted maximum spanning star forest problem (MSSF) is to find a forest with as many edges such that each tree in the forest is a star. The edge-weighted MSSF (eMSSF) is the natural generalization with weights on edges. The node-weighted MSSF (nMSSF) has weights on vertices and the weight of a star is the weight

on the leaves. If the star is just an edge, then the weight of the star is the maximum of the weights of the end points.

It is clear that the non-trivial part of the above problem is to identify the vertices which are the centers of the stars. Once more, we reduce MAX-3-LIN(2) to both eMSSF and nMSSF. Let us discuss the edge-weighted MSSF first. For every variable x in an instance of MAX-3-LIN(2), we introduce two vertices: $\langle x:0\rangle$, $\langle x:1\rangle$. The interpretation is clear: we will enforce that exactly one of these vertices will be the center which will coincide with the assignment in the MAX-3-LIN(2) instance. Such an enforcing is brought about by putting a heavy cost edge between the vertices. For every equation we will add vertices as we did in the reduction to GAP.

In the node-weighted MSSF, we need to add an extra vertex, the *switch vertex*, along with the two vertices $\langle x:0\rangle$, $\langle x:1\rangle$. These vertices form a triangle and have a weight high enough to ensure that exactly one of $\langle x:0\rangle$, $\langle x:1\rangle$ is chosen as a center in any optimum solution to nMSSF.

We remark that Chen et.al [22] also use a similar gadget as the one above, although their reduction is from a variation of MAX-3-SAT and thus their results are weaker.

Hardness of eMSSF: Let I be an instance of MAX-3-LIN(2). Denote the variables as $x_1,\cdots,x_n$. Also let $deg(x_i)$ be the *degree* of variable x_i i.e. the number of equations in which variable x_i occurs. Note that $\sum_i deg(x_i)=3m$. We construct an instance $E(I)$ of eMSSF as follows:

- For every variable x_i, we have two variables which we label as $\langle x_i:0\rangle$ and $\langle x_i:1\rangle$, corresponding to the two assignments. These variables are called variable vertices. There is an edge between them of weight $deg(x_i)/2$.
- For every equation $e: x_i+x_j+x_k=\alpha$ ($\alpha\in\{0,1\}$), we have 4 vertices corresponding to the four assignments to x_i,x_j,x_k which satisfy the equation: $\langle x_i:\alpha,x_j:\alpha,x_k:\alpha\rangle$, $\langle x_i:\alpha,x_j:\bar{\alpha},x_k:\bar{\alpha}\rangle$, $\langle x_i:\bar{\alpha},x_j:\bar{\alpha},x_k:\alpha\rangle$ and $\langle x_i:\bar{\alpha},x_j:\alpha,x_k:\bar{\alpha}\rangle$. This set of vertices are called equation vertices denoted by S_e. Thus we have $4m$ equation vertices in all. Each equation vertex of the form $\langle x_i:\alpha_i,x_j:\alpha_j,x_k:\alpha_k\rangle$ is connected to three variable vertices: $\langle x_i:\alpha_i\rangle$, $\langle x_j:\alpha_j\rangle$ and $\langle x_k:\alpha_k\rangle$. The weight of all these edges is 1. Thus, the degree of every equation vertex is 3 and the degree of every variable vertex $\langle x_i:0\rangle$ or $\langle x_i:1\rangle$ is $2deg(x_i)$.

Lemma 3.4.18 *Given any solution to $E(I)$, there exists a solution of at least the same weight where the centers are exactly one variable vertex per variable.*

Proof: Firstly note that if none of the variable vertices are centers then one can make one of them a center and connect the other to it and get a solution of higher cost (note that the

degrees of the variables in the equations can be assumed to be bigger than 4 by replication). The proof is in two steps. Call a variable $x_i \in I$ *unassigned* if both of $\langle x_i : 0\rangle$ and $\langle x_i : 1\rangle$ is a center. Call a solution of $E(I)$ k-satisfied if exactly k of the variables are assigned. The claim is that there exists a solution of equal or more weight which is n-satisfied. We do this via induction.

We show that if a solution to $E(I)$ is k-satisfied with $k < n$,then we can get a solution of at least this weight which is $k+1$-satisfied. Pick a variable x_i which is unassigned. For every equation $e : x_i + x_j + x_k = 0$, say, containing x_i we claim that one can assume of the four equation vertices in S_e, only one is connected to $\langle x_i : 0\rangle$ and $\langle x_i : 1\rangle$. This is because at least one of the two variable vertices corresponding to both x_j and x_k are centers. Suppose these are $\langle x_j : 0\rangle$ and $\langle x_k : 0\rangle$. Now note that of the four vertices in S_e only $\langle x_i : 0, x_j : 1, x_k : 1\rangle$ is not neighboring to either of these centers. The three remaining can be moved to these without any decrease in the weight of the solution and the claim follows.

Thus, we can assume that for every unassigned variable x_i in the k-satisfied solution, one of the two variable vertices $\langle x_i : 0\rangle$ or $\langle x_i : 1\rangle$ (say $\langle x_i : 0\rangle$), is connected to at most $deg(x_i)/2$ equation vertices. Therefore, disconnecting all the equation items connected to $\langle x_i : 0\rangle$, making it a leaf and connecting it to $\langle x_i : 1\rangle$, gives a $k + 1$-satisfied solution of weight at least the original weight. □

Now we get the hardness of eMSSF using the theorem of Håstad.

Theorem 3.4.19 *For any $\epsilon > 0$, it is NP-hard to approximate edge-weighted MSSF to a factor $10/11 + \epsilon$.*

Proof: The proof follows from the following two calculations and Theorem 3.4.2.

- If $OPT(I) \geq m(1 - \delta)$, then the maximum profit of $E(I)$ is at least $5.5m - 4m\delta$. For every variable x_i, if the assignment of x_i is $\alpha \in \{0, 1\}$, make $\langle x_i : \alpha\rangle$ the center. Observe that for every satisfied equation $e : x_i + x_j + x_k = \alpha$, all the four vertices of S_e can be connected to a center. Thus the weight of $E(I)$ is at least $\sum_i deg(x_i)/2 + 4m(1 - \delta) = 5.5m - 4m\delta$.

- If $OPT(I) \leq m(1/2+\eta)$, then the maximum profit of $E(I)$ is at most $5m+m\eta$ From the claim above we can assume for each variable x_i, one of its two variable vertices is a center. This defines an assignment of truth values to the variables and around half of the equations are not satisfied by this assignment. The observation is that for any unsatisfied equation $e : x_i + x_j + x_k = \alpha$, one of the four equation vertices in S_e is not connected to any center. Thus, the total weight of any solution is at most $\sum_i deg(x_i)/2 + 4m(1/2 + \eta) + 3m(1/2 - \eta) = 5m + m\eta$.

□

Hardness of nMSSF: Let I be an instance of MAX-3-LIN(2). The only difference between the instance of nMSSF, $N(I)$, and the eMSSF $E(I)$ is that for every variable $x_i \in I$, along with the variable vertices $\langle x_i : 0\rangle$ and $\langle x_i : 1\rangle$, we have a switch vertex s_i. The three vertices from a triangle and the node-weights of all of them are $deg(x_i)/2$. The rest of the instance of $N(I)$ is exactly like $E(I)$ with the edge-weights being replaced by node-weights of 1 on the equation vertices.

The reason we require the third switch vertex per variable is that otherwise we cannot argue that in any solution to $N(I)$ at least one of the variable vertices should be a center. With the switch vertex, we can argue that this is the case. If none of the variable vertices is a center, then the switch item is not connected to any vertex. Thus making any one of the variable vertices as a center connected to the switch item gives a solution to $N(I)$ of weight at least the original weight.

Lemma 3.4.18 now holds as in the case of $E(I)$ and thus similar to Theorem 3.4.20 we have the following hardness of node-weighted MSSF.

Theorem 3.4.20 *For any $\epsilon > 0$, it is NP-hard to approximate edge-weighted MSSF to a factor $13/14 + \epsilon$.*

Proof: The proof follows from the following two calculations and Theorem 3.4.2.

- If $OPT(I) \geq m(1-\delta)$, then the maximum profit of $N(I)$ is at least $7m - 4m\delta$. For every variable x_i, if the assignment of x_i is $\alpha \in \{0,1\}$, make $\langle x_i : \alpha\rangle$ the center. Connect the switch item and the vertex $\langle x_i : \bar{\alpha}\rangle$ to this center. Observe that for every satisfied equation $e : x_i + x_j + x_k = \alpha$, all the four vertices of S_e can be connected to a center. Thus the weight of $N(I)$ is at least $\sum_i deg(x_i) + 4m(1-\delta) = 7m - 4m\delta$.

- If $OPT(I) \leq m(1/2+\eta)$, then the maximum profit of $N(I)$ is at most $6.5m+m\eta$ From the claim above we can assume for each variable x_i, one of its two variable vertices is a center. This defines an assignment of truth values to the variables and around half of the equations are not satisfied by this assignment. The observation is that for any unsatisfied equation $e : x_i + x_j + x_k = \alpha$, one of the four equation vertices in S_e is not connected to any center. Thus, the total weight of any solution is at most $\sum_i deg(x_i) + 4m(1/2+\eta) + 3m(1/2-\eta) = 6.5m + m\eta$.

□

3.5 Discussion

In this chapter we studied the maximum budgeted allocation problem and showed that the true approximability lies between 3/4 and 15/16. Our algorithms were based on a natural LP relaxation of the problem and we, in some sense, got the best out if it: the integrality

gap of the LP is 3/4. An approach to get better approximation algorithms might be looking at stronger LP relaxations to the problem. One such relaxation is the *configurational LP relaxation* which we describe below.

3.5.1 The configurational LP relaxation

In this relaxation, we have a variable $x_{i,S}$ for every agent i and every subset of items $S \subseteq Q$. The LP is as follows

$$\begin{aligned} & \text{Max} \quad \{\sum_i \sum_{S \subseteq Q} u_i(S) x_{i,S} \\ \text{s.t.:} \quad & \forall i \in A, \quad \sum_{S \subseteq Q} x_{i,S} \leq 1; \\ & \forall j \in Q, \sum_{i, S \subseteq Q : j \in S} x_{i,S} \leq 1; \\ & \forall i \in A, S \subseteq Q, \qquad x_{i,S} \geq 0\} \end{aligned} \tag{9}$$

The first constraint implies that each agent gets at most one subset of items. The second implies that each item is at most one subset. The value generated on giving a subset S to agent i is $u_i(S) = \min(B_i, \sum_{j \in S} b_{ij})$.

Solving the LP: Observe that the LP has exponentially (in n and m) many variables and an obvious question is how to solve the LP. One does so by going to the dual LP which will have exponentially many constraints but only polynomially many variables. Such a trick is now standard first used by Carr and Vempala [16]. The dual of LP 9 is as follows

$$\begin{aligned} & \text{Min} \quad \{\sum_{i \in A} \alpha_i + \sum_{j \in Q} p_j \\ \text{s.t.:} & \forall i \in A, S \subseteq Q, \quad \alpha_i + \sum_{j \in S} p_j \geq u_i(S); \\ & \forall i \in A, \forall j \in Q, \quad \alpha_i, p_j \geq 0\} \end{aligned} \tag{10}$$

Suppose, for the time being, the LP10 has a separation oracle: Given (α_i, p_j) one can say in polynomial time if it is feasible for LP10 or find a subset of agents S with $\alpha_i + \sum_{j \in S} p_j < u_i(S)$. If so, then the ellipsoid algorithm can be used to solve the LP by making a polynomial number of queries to the separation oracle. Moreover, the subsets returned by the separation oracle are enough to describe the optimal solution to the dual. In other words, in the primal LP 9, only the variables corresponding to these constraints need be positive and the rest can be set to 0 and the optimum is not changed. Since they are only polynomially many, LP 9 can now be solved in polynomial time. Moreover, if one had an r-separation oracle ($r \leq 1$): Given (α_i, p_j) one can say in polynomial time if $\frac{1}{r} \cdot (\alpha_i, p_j)$ is feasible for LP10 or

find a subset of agents S with $\alpha_i + \sum_{j \in S} p_j < u_i(S)$; then the above argument can be used to get an r-approximation for the primal LP 9.

Note that the separation problem for LP 10 is precisely the demand oracle problem described Section 3.1.2: given prices p_j to items, for every agent i find a subset of items maximizing $(u_i(S) - \sum_{j \in S} p_j)$ where $u_i(S) = \min(B_i, \sum_{j \in S} b_{ij})$. The problem is NP-hard with a reduction from PARTITION. Given an instance of partition of n integers $b_1, b_2, \cdots, b_n$ and a target integer B, consider the instance of the separation problem with n items having bids b_1 to b_n and prices $b_1/2, \cdots, b_n/2$ and budget B. Note that the maximum value of the separation problem is at most $B/2$ and moreover the optimum is $B/2$ if and only if there is a subset of the integers adding exactly to B.

We now demonstrate an $(1-\epsilon)$-separation oracle for LP 10 using the FPTAS for the knapsack problem. The sketch below is not the fastest implementation as we interested mainly in the existence of polynomial time algorithms. In the knapsack problem, we are given a capacity of B and n items having profits p_j and weight w_j and the goal is to obtain a subset of items having maximum profit with the total weight being less than B. The problem in NP-hard, but an FPTAS exists. Moreover, there is an exact algorithm which runs in time $O(n^2 P)$ where P is the largest profit. Given the separation problem for LP 10, we consider the items in any arbitrary order. The first item has a bid of b_1 and price p_1. Suppose the item is picked in the optimum solution, call it S. If so, then $\sum_{j \in S} b_{ij} \leq B + b_1$, as otherwise one could discard the first item and get a better solution. Thus the optimum solution of the separation problem given the first item is picked is precisely

$$\max_{0 \leq x \leq b_1} \{\text{Knapsack}[\{(b_2 - p_2, b_2), \cdots, (b_n - p_n, b_n)\}, B - x] + (x - p_1)\}$$

$(x - p_1)$ is the value of the item 1 after the items from $b_2, \cdots, b_n$ use up $B - x$ of the budget. One can repeat the procedure n times removing one item at a time and in the end taking the best solution over all iterations. This gives the optimum solution in time $O(n^3 B^2)$, where B is the budget.

To make the above algorithm run in polynomial time, we round down to the nearest integer all the budgets, bids and prices by a factor of $\epsilon B/n$. The solution to this reduced instance can be found in time $O(n^5/\epsilon)$ and the same solution, scaled back, can be shown to be within $(1-\epsilon)$ of the optimal solution (see for example, [93]).

Integrality gap of the configurational LP: In the next theorem we show that the integrality gap of the configurational LP is between 3/4 and 5/6. The lower bound follows basically by showing that the value of LP 9 is at most the value of LP 6 (and thus is a better upper bound on the optimum). The upper bound follows from an example which we demonstrate below.

Theorem 3.5.1 *The integrality gap of the configurational LP of MBA is between* 3/4 *and* 5/6.

Proof: An easy way to see that the configurational LP is stronger than LP(6) is by looking at the duals of both LP's. One can show that any solution (α_i, p_j) to LP(7) corresponds to a feasible solution $(B_i\alpha_i, p_j)$ to LP 10 of equal value. Thus the configurational LP value is smaller than that of LP(6).

The 5/6-example is as follows: The instance consists of 4 agents a_1, b_1, a_2, b_2. a_1, a_2 have a budget of 1, b_1, b_2 have a budget of 2. There are five items: c, x_1, y_1 and x_2, y_2. Only b_1 and b_2 bid on c and bid 2. For $i = 1, 2$, a_i and b_i each bid on x_i and y_i, and the bid is 1.

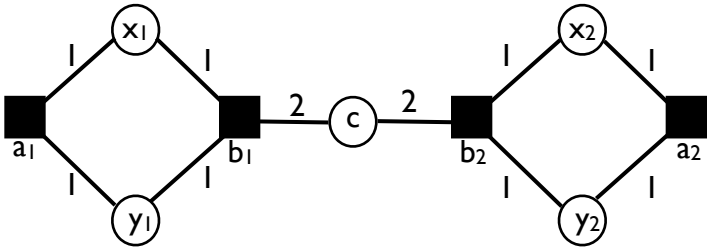

Figure 8: Integrality gap example for configurational LP.

Once again, if c is given to b_1, then either a_2 or b_2 ends up spending 1 less than his budget. Thus, the optimum MBA solution is 5. But there is a solution to the configurational LP(9) of value 6. The sets are $S_1 = \{x_1\}$, $S_2 = \{y_1\}$, $S_3 = \{x_2\}$, $S_4 = \{y_2\}$, $S_5 = \{c\}$,$S_6 = \{x_1, y_1\}$ and $S_7 = \{x_2, y_2\}$. The solution is: $x_{a_1,S_1} = x_{a_1,S_2} = x_{a_2,S_3} = x_{a_2,S_4} = 1/2$ and $x_{b_1,S_6} = x_{b_1,S_5} = x_{b_2,S_5} = x_{b_2,S_7} = 1/2$. □

The above theorem is about all we know for the configurational LP relaxation for MBA. We believe that the integrality gap should be strictly better (larger) than 3/4 although it is not clear how to do so. Configurational LPs have been used for other allocation problems; in fact the best known approximation algorithm of Feige and Vondrák [35] for SMW and GAP proceeds by rounding the solution of the LP. However, we do not know how to use the "simplicity" of the submodular functions of MBA to get a better bound. (Feige and Vondrák get a factor strictly bigger than $1 - 1/e$). We leave the question of pinning down the exact integrality gap of this LP as an open question and believe the resolution might require some new techniques.

CHAPTER IV

ONLINE ALLOCATION PROBLEMS WITH APPLICATIONS TO BUDGETED AUCTIONS

In this chapter we consider the *online version* of a few allocation problems. The most basic allocation problem we consider is KNAPSACK: Given a knapsack (bin) of capacity of B and m items having profits p_j and weight w_j, the goal is to obtain a subset of items having maximum profit with the total weight being at most B. The most general allocation problem which generalizes KNAPSACK we consider is GAP: Given a set A of n knapsacks (or bins) of capacity $B_1, \cdots, B_n$, a set of m items Q, with item j having a profit p_{ij} and weight w_{ij} for bin i, the goal is to find an allocation of items to bins such that the total weight in each bin does not exceed the capacity and the total profit across bins is maximized.

We devise algorithms for online versions of these allocation problems. That is, the set of items Q is not known at the beginning and items arrive one at a time. At each instant, the algorithm must decide what to do with the item (place it in a bin or discard it), and the decision once made is irrevocable - discarded items cannot be recalled back and items once allocated cannot be de-allocated. Nevertheless, the decisions can be made based on items seen so far.

We analyze the performance of our algorithms using competitive analysis. That is, given any set of items J, we compare the profit obtained by our algorithms to the best profit that could have been obtained by *any* (computationally all-powerful) "offline" algorithm who has complete knowledge about the set of items.

Comparing with such an all powerful benchmark has its difficulties - as we make clear later, if one makes no assumptions then no competitive algorithms are possible. We make the following two (necessary) assumptions throughout the chapter:

1. The weights of each item is much smaller than the capacity of the bins. In particular, there is an $\epsilon > 0$ very close to 0, such that $w_{ij}/B_i \leq \epsilon$ for all $i \in A, j \in Q$.

2. The profit-to-weight ratios are neither too high, nor too small. That is, there exists $U, L > 0$ such that for all $i \in A, j \in Q$,

$$L \leq \frac{p_{ij}}{w_{ij}} \leq U$$

Our main results are a $(\ln(U/L) + 2)$-competitive algorithm for online GAP and the online multiple choice knapsack problem (ONLINE-MCKP) and a $(\ln(U/L) + 1)$-competitive algorithm for the single and multiple knapsack problem. We also show these algorithms

are almost optimal. For the simplest case of the single knapsack problem, we show no algorithm can achieve a competitive ratio better than $(\ln(U/L)+1)$. All our algorithms are deterministic.

Relation to budgeted auctions: Apart from being natural questions in the field of online algorithms, the online versions of allocation problems have applications in auctions where the supply is unknown and arbitrary.

Consider the auction of many copies of a single item (say license to some computer software) although the total number of copies is unknown. The auctions runs over time and budget-constrained bidders are allowed to change their bids over time. At each instant a copy of the item arrives, the current bids of bidders are looked at and a second-price auction occurs: the highest bidder is given the item and charged the bid of the second highest one. We argue that a fixed single bidder who can see the bids of the other bidders (this may sometime be possible and, as we will see, sometimes unnecessary), is solving an online knapsack problem. At any point of time before arrival of a new item, the bidder has to decide whether to bid the highest or not - in the former case he gets a profit of the value of the item and his budget depletes by the second highest bid (which corresponds to picking an item), in the latter case he makes none and loses no money (which corresponds to the discard of an item).

Online allocation problems are also useful in designing auctions when the supply is unknown. For instance, consider the auction scenario before, except that there is not a single item but many such items. Moreover, suppose giving an item j to bidder i gives a profit of p_{ij} while it depletes the bidder's budget by w_{ij} = the bid on the item. Thus the online version of GAP can be used to obtain a socially efficient [1] first-price auction of these items. If one considers only the profit of the auctioneer instead, then the design of the optimum auction is precisely the special case of online GAP with $p_{ij} = w_{ij}$.

The most famous (and financially significant) such unknown supply auctions are the contemporary sponsored search auctions hosted by Internet search engines. In these auctions, bidders, who are advertisers, bid on keywords (like "`color printers`" or "`turkish towels`") and each time this keyword is queried by some user via the search engine, the auction takes place. Thus the total supply of the items (queries of keywords) is unknown. We discuss these auctions in a little more detail in Section 4.4.

4.1 The online knapsack problem

The online knapsack problem was first studied by Marchetti-Spaccamela and Vercellis [75]. Indeed, they were the first to show that no non-trivial competitive algorithms existed for

[1] An auction is called efficient if the total utility of all the agents (bidders and auctioneer) is maximized. An auction is called optimal if it maximizes the revenue to the auctioneer

the general case of the online knapsack problem. To see this, consider the case where the capacity $B = 1$ and consider two input sequences both having two items, $\sigma_1 = \{(1,1),(0,1)\}$ and $\sigma_2 = \{(1,1),(\infty,1)\}$. Its easy to see any deterministic online algorithm will be infinitely worse than the omniscient algorithm on *at least one* of the inputs. It is not too hard to generalize this argument to randomized online algorithms as well.

Besides giving the impossibility result, [75] studied the online knapsack problem in the average case setting with the profits and weights of the items being picked from a fixed distribution. They gave an online algorithm which in this setting gave a profit an additive factor away from the optimal. Lueker [73] improved the additive factor and gave an optimal algorithm in this setting. Other variants of the stochastic knapsack problems were considered by Papastavrou et.al [81, 62] and Van-Slyke and Young [92]. Other variants of the online knapsack problem like the removable online knapsack problems [50] and online partially fractional knapsack problems [80] have also been studied recently.

As far our knowledge, we do not know of any work on the online knapsack problem with the assumptions that we make. However, recently, in a series of works Buchbinder and Naor [14, 15] designed online algorithms for fractional versions of general packing problems, our allocation problems are also packing problems. One can derive $O(\ln(U/L))$-competitive online algorithms for our problems using their framework. Nevertheless, we believe our algorithms are much simpler for the special case we consider and furthermore our results are optimal or off by an additive $+1$.

In this section we present two $(\ln(U/L)+1)$-competitive algorithms for the online knapsack problem. The first is an extremely simple randomized online algorithm and works only when the input sequence of items is *oblivious*, that is, the item coming next doesn't depend on the choices of the algorithm. The second is a deterministic (although it is not a derandomization of the above) online algorithm which works with adaptive sequences of items as well.

Given an item j, we call the ratio of the profit p_j of the item to its weight w_j the *efficiency* of the item. The second assumption which we work on is that the efficiency of an item is between L and U.

Randomized Algorithm: Let $\mathcal{D}$ be the following continuous distribution from 0 to U, with the density function $f(x) = \frac{c}{x}$, for $L \le x \le U$, and $f(x) = c/L$ for $0 \le x \le L$, where $c = \frac{1}{1+\ln(U/L)}$ Note that $\int_0^U f(x)dx = 1$, and this is a valid density function.

Algorithm 10 ONLINE-KP-RANDOMIZED

Pick a threshold T from the distribution $\mathcal{D}$.
Whenever item j arrives, pick it iff

$$\frac{p_j}{w_j} \ge T \qquad \text{and capacity remaining} \ge w_j.$$

Theorem 4.1.1 ONLINE-KP-RANDOMIZED *has randomized competitive ratio* $(\ln(U/L) + 1)$.

Proof: Let σ be an input sequence of items and $\mathcal{A}(\sigma)$ be the profit obtained by the algorithm above. $\mathbf{E}[\mathcal{A}(\sigma)]$ is the expected profit, with the expectation over the random choice of the threshold T.

Suppose that the optimum offline fills the knapsack up to capacity $\alpha \cdot B$, for some $\alpha \leq 1$. Given an input sequence of items σ, for $x \in [0, U]$, let $\rho(x)$ denote the fraction of the knapsack filled by the optimum offline algorithm with items whose efficiency ratio is more than x. Note that ρ is an integrable[2] decreasing function with $\rho(0) = \rho(L) = \alpha$ and $\rho(U) = 0$.

Also observe that the profit obtained by the optimum offline algorithm is given by:

$$\mathrm{OPT}(\sigma) = \int \mathrm{x} \cdot \mathrm{B}(-\mathrm{d}(\rho(\mathrm{x}))) = \int_0^{\mathrm{U}} \rho(\mathrm{x})\mathrm{dx} \cdot \mathrm{B} \tag{11}$$

The first integral is written with an abuse of notation: it is actually the sum of integrals with the same integrand with ranges where ρ is continuous. The second equality follows from simple calculus.

Now note that if the random threshold chosen is $T \geq L$, then the algorithm would have a profit of at least $T\rho(T) \cdot B$ as there are items which fill at least $\rho(T) \cdot B$ and have efficiency more than T. Also if $0 \leq T \leq L$, then the algorithm would have a profit of at least $L \cdot B$. (Actually, we have used assumption 1 ($w_j \ll B$); refer to the remark after the proof.)

Thus the expected profit of the algorithm

$$\begin{aligned}
\mathbf{E}[\mathcal{A}(\sigma)] &\geq \int_0^L LBf(x)dx + \int_L^U x\rho(x)Bf(x)dx \\
&= LB\int_0^L \frac{c}{L}dx + B\int_L^U x\rho(x)\frac{c}{x}dx \\
&\geq cB\int_0^L \rho(x)dx + cB\int_L^U \rho(x)dx \\
&= c \cdot \mathrm{OPT}
\end{aligned}$$

where the last but one inequality uses $\rho(x) = \alpha \leq 1$ for $x \in [0, L]$. The proof follows by observing $c = \frac{1}{\ln(U/L)+1}$. □

Remark: Note that we are use the fact that the weights are much smaller than the capacity when we say the algorithm fills $T\rho(T) \cdot B$ portion of the knapsack with items of efficiency more than T. To be precise, with assumption 1, we would say the algorithm has a profit at least $T\rho(T) \cdot B(1 - \epsilon)$, since the algorithm is guaranteed to fill $\rho(T) \cdot B(1 - \epsilon)$ capacity with items with efficiency more than T. Thus, to be precise, the competitive ratio is

[2] In fact, ρ is a step function with a finite number of steps

$\frac{1}{(1-\epsilon)}(\ln(U/L)+1)$.

Remark: Note that the algorithm works well only in expectation and against oblivious adversaries. If the threshold choice is known, then an adversary can produce an arbitrarily bad input sequence.

Deterministic Algorithm: Now we present the deterministic algorithm for the online knapsack problem which works against all adversaries achieving the optimal bound of $\ln(U/L)+1$. In the remainder of the chapter, e denotes the base of the natural logarithm.

Algorithm 11 ONLINE-KP-THRESHOLD

Let $\Psi(z) \equiv (Ue/L)^z(L/e)$.
When item j arrives, let z_j be the fraction of capacity filled, pick element j iff

$$\frac{p_j}{w_j} \geq \Psi(z_j).$$

Observe that for $z \in [0,c]$ where $c \equiv \frac{1}{1+\ln(U/L)}$, $\Psi(z) \leq L$, thus the algorithm will pick all items available until c fraction of the knapsack is filled. In fact, we will assume henceforth $\Psi(z) = L$ for $z \in [0,c]$. When $z = 1$, $\Psi(z) = U$, and since Ψ is strictly increasing, the algorithm will never over-fill the knapsack.

Theorem 4.1.2 ONLINE-KP-THRESHOLD *has a competitive ratio of* $\ln(U/L)+1$.

Proof: Fix an input sequence σ. Let the algorithm terminate filling Z fraction of the knapsack and obtaining a value of $\mathcal{A}(\sigma)$. Let S and S^* respectively be the set of items picked by the Algorithm ONLINE-KP-THRESHOLD and the optimum. Denote the weight and the value of the common items by $W = \sum_{j\in(S\cap S^*)} w_j$ and $P = \sum_{j\in(S\cap S^*)} p_j$. For each item j not picked by the algorithm, its efficiency is $< \Psi(z_j) \leq \Psi(Z)$ since $\Psi(z)$ is the monotone increasing function of z. Thus,

$$\mathrm{OPT}(\sigma) \leq \mathrm{P} + \Psi(\mathrm{Z})(\mathrm{B} - \mathrm{W})$$

Since $\mathcal{A}(\sigma) = P + \sum_{j\in(S\setminus S^*)} p_j =: p(S \setminus S^*)$, the above inequality implies that

$$\frac{\mathrm{OPT}(\sigma)}{\mathcal{A}(\sigma)} \leq \frac{P + \Psi(Z)(B-W)}{P + p(S \setminus S^*)}. \tag{12}$$

Since each item j picked in S must have efficiency at least $\Psi(z_j)$ where z_j is the fraction

of the knapsack filled at that instant, we have

$$P \geq \sum_{j \in S \cap S^*} \Psi(z_j) w_j =: P_1, \tag{13}$$

$$p(S \setminus S^*) \geq \sum_{j \in S \setminus S^*} \Psi(z_j) w_j =: P_2. \tag{14}$$

Since $\mathrm{OPT}(\sigma) \geq \mathcal{A}(\sigma)$, inequality (12) implies

$$\frac{\mathrm{OPT}(\sigma)}{\mathcal{A}(\sigma)} \leq \frac{P + \Psi(Z)(B - W)}{P + p(S \setminus S^*)} \leq \frac{P_1 + \Psi(Z)(B - W)}{P_1 + p(S \setminus S^*)} \tag{15}$$

Noting that $P_1 \leq \Psi(Z) w(S \cap S^*) = \Psi(Z) W$ and plugging in the values of P_1 and P_2 we get

$$\frac{\mathrm{OPT}(\sigma)}{\mathcal{A}(\sigma)} \leq \frac{\Psi(Z)}{\sum_{j \in S} \Psi(z_j) \Delta z_j} \tag{16}$$

where $\Delta z_j = z_{j+1} - z_j = w_j / B$ for all j.

Now based on the assumption that the weights are much smaller than B, we can approximate the summation via an integration (refer to the remark following the proof). Thus,

$$\begin{aligned}
\sum_{j \in S} \Psi(z_j) \Delta z_j &\approx \int_0^Z \Psi(z) dz \\
&= \int_0^c L dz + \int_c^Z \Psi(z) dz \\
&= cL + \frac{L}{e} \frac{(Ue/L)^Z - (Ue/L)^c}{\ln(Ue/L)} \\
&= \frac{L}{e} \frac{(Ue/L)^Z}{\ln(Ue/L)} = \frac{\Psi(Z)}{\ln(U/L) + 1}
\end{aligned}$$

and along with inequality (16) this completes the proof. □

Remark: Just as in Theorem 4.1.1, we can make the approximation made above precise. Since $\Psi(z)$ is an increasing function of z, we obtain $\sum_{j \in S} \Psi(z_j) \Delta z_j \geq (1 - \epsilon) \int_0^Z \Psi(z) dz$ where $\epsilon = (\max_j w_j)/B$ is a small constant. Thus, to be precise, the competitive ratio is actually $\ln(Ue/L) \cdot \frac{1}{1-\epsilon}$.

4.1.1 Extension to online multiple knapsack problem

In the multiple knapsack problem, as the name suggests we have n knapsacks or bins, each having a capacity of $B_1, \cdots, B_n$. However, the profit and weights of items are independent of the bins.

The offline version of the multiple knapsack problem is harder than the single knapsack problem (the latter has an FPTAS while the former has only a PTAS[21] and does not

have an FPTAS unless P=NP). The hardness arises from the fact that even if the set of items to be picked is known, to distribute among the different knapsacks is an instance of bin-packing. However, when we make assumption 1, that is, the weights of items are much smaller than the capacity of the knapsack, this issue of selecting bins doesn't apply any more and thus the multiple knapsack problem is in some sense as easy as the single knapsack problem.

In fact, the algorithm for the online multiple knapsack problem is the same as Algorithm 11 with an item j picked if its efficiency is larger than the $\Psi()$ of some bin (ties broken arbitrarily). The proof of the following theorem is very similar to the proof of Theorem 4.1.2.

Theorem 4.1.3 ONLINE-KP-THRESHOLD *with the above modification is a* $(\ln(U/L)+1)$*-competitive algorithm for the online multiple knapsack problem.*

Proof: Let σ be a fixed input sequence and let the algorithm terminate filling $Z_1, Z_2, \cdots, Z_n$ fraction respectively of the n knapsacks obtaining a value of $\mathcal{A}(\sigma)$. Let S be the set of items picked by the algorithm and let $S_1, S_2, \cdots, S_n$ be the natural partition of S: S_i is set of items the algorithm picks in the ith knapsack.

Let $W_i := \sum_{j\in(S_i\cap S^*)} w_j$ and $P_i = \sum_{j\in(S\cap S^*)} p_j$. As before we have that for each item j not picked by the algorithm, its efficiency is $\leq \Psi(Z_i)$, for all i. Thus we get

$$\frac{\mathrm{OPT}(\sigma)}{\mathcal{A}(\sigma)} \leq \frac{\sum_{i=1}^{n}(\ P_i + \Psi(Z_i)(B_i - W_i)\ )}{\sum_{i=1}^{n}(\ P_i + p(S_i \setminus S^*)\ )}.$$

As in the proof of Theorem 4.1.2, we have for all i, $P_i \geq \sum_{j\in S_i\cap S^*} \Psi(z_{ij})w_j$ where z_{ij} is the fraction of the capacity of the ith knapsack filled when item j arrives. Thus,

$$\frac{\mathrm{OPT}(\sigma)}{\mathcal{A}(\sigma)} \leq \frac{\sum_{i=1}^{n}(\ \sum_{j\in S_i\cap S^*} \Psi(z_{ij})w_j + \Psi(Z_i)(B_i - W_i)\ )}{\sum_{i=1}^{n}(\ \sum_{j\in S_i\cap S^*} \Psi(z_{ij})w_j + p(S_i \setminus S^*)\ )} \tag{17}$$

The numerator is less than $\sum_{i=1}^{n}(\ \Psi(Z_i)W_i + \Psi(Z_i)(B_i - W_i) = \sum_{i=1}^{n} \Psi(Z_i)B_i$.
The denominator is equal to $\sum_{i=1}^{n}(\ \sum_{j\in S_i} \Psi(z_{ij})B_i \cdot \Delta z_{ij}\ )$, where $\Delta z_{ij} := z_{i,j+1} - z_{ij} = w_j/B_i$.

As in the proof of Theorem 4.1.2, each of the ratios $(\Psi(Z_i)/\sum_{j\in S_i} \Psi(z_{ij})\Delta z_{ij})$ is less than $\frac{1}{(\ln(U/L)+1)}$ implying the RHS of inequality 17 is less than $\frac{1}{(\ln(U/L)+1)}$ completing the proof. □

4.2 Extensions to online multiple choice knapsack problem and online GAP

The online multiple choice knapsack problem (MCKP) is a generalization of the online single knapsack problem. At each time instant t, a set of items N_t arrives and the algorithm

has to choose *at most* one item from the set. The goal again is to get as high a profit without violating the capacity constraint of the knapsack. The algorithm for online MCKP is a simple modification of the online single knapsack problem: for every set N_t, find the elements which cross the threshold as per the online knapsack algorithm, and choose the one with maximum profit. The analysis argument as shown in Theorem 4.2.1 follows the argument of the online knapsack problem, except for the times when the algorithm picks one item from a set and the optimum picks another. Since the algorithm picks the more profitable of the two, these items can be "paid for" by the extra +1 in the competitive factor.

Algorithm 12 ONLINE-MCKP-THRESHOLD

Let $\Psi(z) \equiv (Ue/L)^z(L/e)$.
At time t, let $z(t)$ denote the fraction of capacity filled,

$$E_t \equiv \left\{ s \in N_t \mid \frac{p_s}{w_s} \geq \Psi(z(t)) \right\},$$

pick element $s \in E_t$ with maximum p_s

The above algorithm has a competitive ratio of $\ln(U/L) + 2$, stated as the following theorem.

Theorem 4.2.1 *Algorithm* ONLINE-MCKP-THRESHOLD *has a competitive ratio of* $(\ln(U/L)+2)$.

Proof: For any input sequence of sets σ, let $\mathcal{A}(\sigma)$ be the profit obtained by the above algorithm and $\mathrm{OPT}(\sigma)$ be the maximum profit obtainable. We claim that for any σ,

$$\mathrm{OPT}(\sigma) - \mathcal{A}(\sigma) \leq (\ln(\mathrm{U/L}) + 1)\mathcal{A}(\sigma).$$

Note that the claim immediately implies the theorem. As in the proof of Theorem 4.1.2, let S and S^* be the set of items picked by the algorithm and the optimum, respectively. Let $P = \pi(S \cap S^*)$ denote the profit of the common items, $W = w(S \cap S^*)$ denote the weight. As before, we want to bound the profit of the items picked by OPT but not by ALG. In the multiple-choice case, unlike in the proof of Theorem 4.1.2, the efficiency of an item selected by OPT from N_t is not necessarily bounded by $\Psi(z(t))$ since ALG may have also selected a different item from N_t. Thus we partition the items picked by OPT and not by ALG into two: items which do not satisfy the efficiency condition, and the items which do. Thus the first kind of items have efficiency less than $\Psi(z(t))$, while for the second kind of items, the total profit of these items is less than $\mathcal{A}(\sigma)$ since ALG picks the most profitable item from the same set which satisfy the efficiency condition. We can exclude the second types of items from further consideration since they in total result in at most a profit of $\mathcal{A}(\sigma)$. Now we can assume that all items have efficiency $< \Psi(z(t))$ at time t, thus it returns to a

similar situation as in the proof of Theorem 4.1.2. A similar proof shows that the above claim holds. □

In the online GAP problem, we have n bins with capacities $B_1, \cdots, B_n$ and at each time instant an item j comes with profit p_{ij} and weight w_{ij} for bin i. The goal is to allocate the item to a bin or discard it. The algorithm is very similar to the above algorithm: for each item, pick the bins which satisfy the threshold constraint of the single knapsack algorithm and assign it to the bin to which it gives the highest profit.

Algorithm 13 ONLINE-GAP-THRESHOLD

Let $\Psi(z) \equiv (Ue/L)^z(L/e)$.
Whenever item j arrives, let $z_1, z_2, \cdots, z_n$ be the fraction of the bins filled already.

$$E_j \equiv \left\{ i \in [n] \mid \frac{p_{ij}}{w_{ij}} \geq \Psi(z_i) \right\},$$

Allocate item j to the bin i in E_j with maximum p_{ij}.

Theorem 4.2.2 *Algorithm* ONLINE-GAP-THRESHOLD *has a competitive ratio of* $(\ln(U/L)+2)$ *for online GAP.*

Proof: Fix an input sequence σ of the item set Q. Suppose the algorithm ONLINE-GAP-THRESHOLD fills the bins to capacity $Z_1, Z_2, \cdots, Z_n$. Let $S_i, S_i^* \subset Q$ be the subset of items allocated by the algorithm and optimum respectively to bin i. Also let $S := \cup_i S_i$ and $S^* := \cup_i S_i^*$.

We partition the set $S_i^* \setminus S$ into X_i^* and Y_i^*. X_i^* contains all items j which are not picked in S_i since its efficiency is less than $\Psi(z_{i,j}) < \Psi(Z_i)$, z_{ij} be the fraction of the ith bin filled when the jth item arrived. Y_i^* are all those elements which satisfy the efficiency condition but is allocated to a different bin since it gives more profit in that bin. Thus for all i, the items in Y_i^* are allocated by the algorithm but not in bin i.

Observe that the value obtained by the algorithm on allocating items of Y_i^* (call it $p_i(Y_i^*)$) is *more* than that obtained by the optimum algorithm. This is because the algorithm had an option of allocating these items to the bin to which the optimum algorithm allocated (that is i) but chose a more profitable bin instead. Thus,

$$\sum_{i=1}^{n} p_i(Y_i^*) \leq \sum_{i=1}^{n} p_i(S_i) \tag{18}$$

Now as in the proof of Theorems 4.1.2 and 4.2.1, we have the following bound on the remaining items picked by the optimum algorithm,

$$\frac{\sum_{i=1}^{n}(p_i(S_i^*) - p(Y_i^*))}{\sum_{i=1}^{n} p_i(S_i)} \leq \frac{\sum_{i=1}^{n} \Psi(Z_i)B_i}{\sum_{i=1}^{n}(\ \sum_{j \in S_i} \Psi(z_{ij})B_i \cdot \Delta z_{ij}\ )} \leq (\ln(U/L) + 1)$$

Along with the inequality 18, we get the theorem. □

Remark: An interesting special case which was noted in the introduction is the special case of designing optimum auctions. In such cases, the profit (to the auctioneer, here) p_{ij} and the weight w_{ij} both equal to the bid of bidder i on item j. Note that the algorithm ONLINE-GAP-THRESHOLD reduces to the following greedy algorithm: give item to the highest bidder and budget remaining. The competitive ratio of this greedy algorithm is known to be 2 and this is what is implied by theorem 4.2.2.

4.3 A matching lower bound

In this section we use Yao's minimax technique [96] to get a lower bound on the competitive ratio of online knapsack problem, matching the upper bound given in Theorem 4.1.2.

Theorem 4.3.1 *The competitive ratio of any (possibly randomized) online algorithm for the online knapsack problem is at least* $(\ln(U/L) + 1)$.

Proof: Yao's minimax principle says for any input distribution $\mathcal{D}$ and any γ-competitive randomized algorithm $\mathcal{A}$,

$$\frac{1}{\gamma} \leq \min_{\sigma} \frac{\mathbf{E}[\mathcal{A}(\sigma)]}{\mathrm{OPT}(\sigma)} \leq \max_{\text{deterministic } A} \mathbf{E}_{\sigma \leftarrow \mathcal{D}} \left[\frac{A(\sigma)}{\mathrm{OPT}(\sigma)} \right]$$

To prove the theorem we specify a distribution $\mathcal{D}$ such that

$$\max_{\text{deterministic } A} \mathbf{E}_{\sigma \leftarrow \mathcal{D}} \left[\frac{A(\sigma)}{\mathrm{OPT}(\sigma)} \right] \leq \frac{1}{\ln(U/L) + 1}. \tag{19}$$

Fix a parameter $\eta > 0$. Let k be the largest integer such that $(1+\eta)^k \leq U/L$, i.e., $k = \lfloor \frac{\ln(U/L)}{\ln(1+\eta)} \rfloor$

The support of the input distribution $\mathcal{D}$ consists of the instances $I_0, I_1, \cdots, I_k$, where I_0 is a stream of B identical items each with weight 1 and value L. I_1 is I_0 followed by a stream of B identical items each with weight 1 and value $(1+\eta)L$, and in general I_{j+1} is I_j followed by B items with weight 1 and value $(1+\eta)^{j+1}L$. The distribution $\mathcal{D}$ is specified by giving probability p_j to instance I_j (we specify p_j's later).

Given knowledge of this distribution, any deterministic algorithm A can be fully specified by the vector $(f_0, f_1, \cdots, f_k)$, where f_i is the fraction of the knapsack it fills with items having efficiency ratio $(1+\eta)^i L$. Note that the f_j's could depend on the p_j's. Also note that $\sum_{j=0}^{k} f_j \leq 1$.Thus we have

$$\mathbf{E}_{\sigma \leftarrow \mathcal{D}} \left[\frac{A(\sigma)}{\mathrm{OPT}(\sigma)} \right] = \sum_{i=0}^{k} \frac{p_i \sum_{j=0}^{i} (1+\eta)^j f_j}{(1+\eta)^i} = \sum_{j=0}^{k} f_j \sum_{i=j}^{k} p_i (1+\eta)^{j-i}$$

Now we specify the p_j's, $p_k := \frac{1+\eta}{(k+1)\eta+1}$ and $p_0 = p_1 = \cdots = p_{k-1} := \frac{\eta}{(k+1)\eta+1}$. Note that the p_j's sum to 1. Let $X = (k+1)\eta + 1$. For any j,

$$\begin{aligned}
\sum_{i=j}^{k} p_i(1+\eta)^{j-i} &= p_k(1+\eta)^{j-k} + \sum_{i=j}^{k-1} p_i(1+\eta)^{j-i} \\
&= \frac{(1+\eta)^{j-k+1}}{X} + \frac{\eta}{X}\sum_{i=j}^{k-1}(1+\eta)^{j-i} \\
&= \frac{(1+\eta)^{j-k+1}}{X} + \frac{\eta}{X}\left(\frac{(1+\eta) - (1+\eta)^{j-k+1}}{\eta}\right) \\
&= \frac{1+\eta}{X} = \frac{1+\eta}{(k+1)\eta+1}
\end{aligned}$$

Thus we get

$$\mathbf{E}_{\sigma \leftarrow \mathcal{D}}\left[\frac{A(\sigma)}{\mathrm{OPT}(\sigma)}\right] = \frac{1+\eta}{(k+1)\eta+1}\sum_{j=0}^{k} f_j \le \frac{1+\eta}{(k+1)\eta+1} \le \frac{(1+\eta)}{\eta\frac{\ln(U/L)}{\ln(1+\eta)}+1}$$

where the inequality uses the fact that $\sum_{i=0}^{k} f_i \le 1$ as the algorithm can not over-fill the knapsack. The proof completes by setting $\eta \to 0$ and noting that $\frac{\eta}{\ln(1+\eta)} \to 1$. □

4.4 Applications to sponsored search auctions

Sponsored search auctions hosted by Internet search engines like Google, Yahoo! and MSN to name a few, are a rapidly booming market generating billions of dollars of revenue annually for the search companies. For any given keyword, hundreds of bidders bid on it and are allowed to dynamically revise their bids. We assume there is a minimum bid b_{min} to stay in the auction (usually it is 10 cents). At any moment of time a query is made for the keyword, the search engine allocates the highest S bidders (b_1 to b_S, say) to the S slots and displays their advertisements. The click-through-rate (CTR) of a slot s (probability an advertisement on slot s is clicked), denoted as $\alpha(s)$, [3] is empirically assumed to decrease from upper to lower slots. If an advertisement on slot s is clicked, the advertiser is charged the bid of the bidder b_{s+1}.

Suppose we want to devise a bidding strategy for an advertiser in such an auction. Advertisers associate a revenue-per-click (also called value), which we assume to be V for the advertiser in question (we call him bidder 0 henceforth). We also assume a budget B over a time period T (say 24 hours if the budget is daily). The budget constraint is hard in the sense once exhausted its not refilled and budget remaining at the end of T is taken away.

[3] To be precise, the CTR of a slot depends on the advertiser, but we assume its the same for all only for economy of notation.

We discretize the time T into periods $\{1, 2, \cdots, T\}$ so that no bidder changes his bid in the time interval $[t, t+1)$. Let $X(t)$ denote the expected number of queries for the keyword in this interval. Thus, if at time t bidder 0 bids $b_0(t)$ and gets slot s, the expected revenue he makes in the time slot is $VX(t)\alpha(s)$. Assume the bidder next to him also bids the same amount. Thus bidder 0 pays $b_0(t)X(t)\alpha(s)$.Our problem can be stated thus: At each time t, bid $b_0(t)$, so as to maximize revenue keeping total cost within budget.

It is not too hard to see that if bids of all the agents at each time period is known (by an omniscient bidder), then the best bidding strategy corresponds to solving a knapsack problem (This observation was also made by [11].) Consider the single slot case first. Let $b(t)$ denote the highest bid of other bidders at time t. Thus the problem for the omniscient bidder translates to just deciding which time periods to outbid. This corresponds to picking a subset of items from T having values $\pi(t)$ and weights $w(t)$ with total weight constrained, which is precisely the classic knapsack problem. In the multiple slot case, the omniscient bidder also has to decide which slot he wants the advertisement in. Since he is constrained to picking at most one slot, this corresponds to the multiple-choice knapsack problem: Given sets of items, pick at most one item from each set to maximize value, keeping total weight constrained by capacity of the knapsack.

Bidder 0 does not know the future, and thus designing bidding strategies for our problem amounts to designing algorithms for online versions of the knapsack problems which we studied above. Indeed, in the case of sponsored search auctions, the two assumptions made for the online knapsack problems can be justified.

Let us first consider the single slot case. As stated above, we are going to use the algorithm for the online knapsack problem developed in Section 4.1 for this case. Firstly we need to check if the assumptions go through. The first assumption translates to bids of agents being much smaller than their budgets, which is a common assumption made in these auctions. The second is about the boundedness of the profit-to-weight ratio of the items. In the case of sponsored search auctions, the profit of an item (query) is the value to the bidder, V. The weight is the money he pays if he wins, that is, b_2, the bid of the second-highest bidder. If one assumes a minimum bid b_{min}, we see that the profit-to-weight ratio is upper bounded by $U = V/b_{min}$.[4] Moreover, one can assume one does not bid more than ones valuation. Thus on the queries in which the bidder wins, we have value more than his bid more than the second bidder's bid. Thus $L = 1$ is a lower bound on the profit-to-weight ratio.

The strategies are derived from the online algorithms: Bidder 0 outbids only if the efficiency

[4]To get an idea of the magnitude, roughly on an average companies value a click at around \$2-\$3 while b_{min} is around 10cents. Thus U is in the order of magnitude of 20 to 30

is bigger than the threshold suggested by the algorithm.Since the threshold does not depend on anything other than the fraction of knapsack filled, the strategies also depend only on the fraction of budget spent and doesn't require to know the parameters of the auction at all. Thus the strategies are oblivious bidding strategies and in fact hold for any single item auction where the assumptions (for the online knapsack problem) are met. The strategies are formally stated as follows:

Bidding Strategy: Revenue-Maximizing Single-Slot
Let $\Psi(z) \equiv (Ve/b_{min})^z(1/e)$.
At time t, if fraction of budget spent is $z(t)$, then bid $b_0(t) = \frac{V}{1+\Psi(z(t))}$

We use Revenue to denote the revenue earned by the strategies and $\mathrm{OPT_r}$ denote the revenue of an omniscient bidder. Then the following theorem follows from Theorem 4.1.2:

Theorem 4.4.1 $\mathrm{OPT_r} \leq \ln\left(\frac{\mathrm{e \cdot V}}{\mathrm{b_{min}}}\right) \cdot \mathrm{Revenue}$.

The extension to multiple slots would use the online multiple choice knapsack problem algorithm presented in Section 4.2. The only difference between the online multiple-choice knapsack algorithm and the online single knapsack algorithm is that the former first filters the items from the set which cross the threshold and then chooses the one with the highest profit. In the sponsored search auction setting this translates to being in a slot which satisfies the threshold condition and gives the highest value. Across slots, assuming click-through-rate, $\alpha(s)$ decreased from top to bottom, higher slots give an advertiser more value. Thus the advertiser should bid as high (as bidding high implies higher slots) as the threshold condition suggests. Once again, the condition depends only on the money spent and thus we get the same oblivious bidding strategy as the single slot case giving the following guarantee.

Theorem 4.4.2 *The bidding strategy for single-slot auctions also gives the following guarantee for multiple-slot auctions:*

$$\mathrm{OPT_r} \;\leq\; (\ln(\mathrm{V/b_{min}}) + 2) \cdot \mathrm{Revenue}$$

4.5 Discussion

In this chapter, we looked at online allocation problems and gave almost optimal (in the sense of competitive ratio) online algorithms. In fact, we give optimal algorithms for online knapsack and online multiple knapsack, while for online GAP and multiple-choice knapsack problems, we are off by an additive factor of 1. One possible question is to close this gap - we believe for these problems one can probably get better algorithms than what we presented.

Our main motivation in studying online allocation problems were to develop bidding strategies in budgeted auctions, particularly the sponsored search auctions. One of the

shortcomings of the bidding strategies that are implied by our algorithms (those presented in Section 4.4) is that on a "typical" instance, say a random instance, the strategies end up not bidding high at all. The reason is that the algorithms are designed with an worst-case adversary in mind while in *real life* (that is real auctions over the Web) this is hardly the case.

A few modifications to the strategies were proposed in [20], the work on which this chapter is based, like sniping at the end of the day, which helped in improving the performance of these strategies in practice (simulations). However, to undertake a more theoretically based study, one needs to develop better models of other bidders and incorporate the game-theoretic elements as well.

CHAPTER V

DESIGN IS AS EASY AS OPTIMIZATION

In this chapter, we undertake a systematic study of max-min and min-max optimization problems subject to a global budget (or weight) constraint. We call such problems *design problems*. Every optimization problem leads to a natural design problem; if the optimization problem is a minimization (maximization) problem, its design version is a max-min (min-max) problem.

The process of obtaining a design problem from an optimization problem is formally defined in Section 5.1. As an illustration, the design problem obtained from the sparsest cut problem is: We are given an undirected graph $G(V, E)$ and a bound B on the total weight. The problem is to find a way to distribute weight B on the edges of G so that the sparsity of the sparsest cut is maximized. Observe that this is a max-min problem.

The history of such problems goes back all the way to Fulkerson [39], who considered the problem of maximizing the minimum cut in a network whose edge capacities could be augmented, given a bound on the total augmentation allowed. Observe that since the minimum cut in a network equals the maximum flow, this max-min problem can be transformed into a pure maximization problem: that of finding the augmented network that supports maximum possible flow.

This brings us to the justification of the name "design problem": Assume that the underlying optimization problem is a minimization problem, i.e., among the set of feasible solutions, which is typically exponentially large, it is seeking a minimum cost solution. Then the corresponding design problem, with a given budget, is seeking an instance (among all instances satisfying the budget constraint) in which the minimum cost solution is as large as possible. Thus the task at hand is to design the best *instance* satisfying certain constraints and properties. With this explanation, it should be clear that design problems arise in numerous applications.

The main result in this chapter is that for a large class of optimization problems, the design version of the problem is as easy to solve as the optimization problem itself. That is if there is an α-approximation to the polynomial time algorithm, then there is an α-approximation to the design problem as well (with possibly some additive error). The class of problems we show this for are minimization problems with concave objective functions, and maximizations problems with convex objective functions. An important special class is the class of problems with linear objective functions. These classes will be defined formally in Section 5.1. The general theorem is derived using the ellipsoid method and thus opens up questions for faster algorithms. For the class of problems with linear objective functions,

we use algorithms from learning theory to obtain faster algorithms.

Related work: As in the work of Fulkerson [39], Jüttner [54] studied a general class of problems in which a transformation of a max-min problem into a purely maximization problem always works; he called it *budgeted optimization problems*. Start with any optimization problem for which the set of feasible solutions forms a polytope. Then, the max-min or min-max problem obtained from it (depending on whether the original problem is a minimization or maximization problem) is in this class. Jüttner showed the general result that if the original optimization problem has a strongly polynomial algorithm, then so does the budgeted optimization problem. A key step in obtaining this result is captured in the solution to Fulkerson's problem. W.l.o.g. assume that the budgeted optimization problem is a max-min problem, which has been obtained from a minimization problem. Now, using fact that the set of feasible solutions of the latter form a polytope, it can be written as a minimization LP. Its dual, a maximization LP, also achieves the same optimal solution, thereby transforming the max-min problem into a max-max problem, which is simply a maximization problem. The latter is solved using Megiddo's parametric search method [76]. Besides Jüttner's work, we are not aware of any systematic study of the complexity of design problems.

In retrospect, Jüttner has carved out a subclass of design problems that, via a polynomial amount of work, can be restated as optimization problems. In fact, this observation was independently made by us and is described in Section 5.2.2. This also has consequences for packing problems which will be the focus of the next chapter. Here we study a more general class of problems in which the underlying optimization problem can have an arbitrary set of feasible solutions, may not even be polynomial time solvable and moreover may not even be linear (but needs to be a concave minimization problem or a convex maximization problem; see Section 5.1). On the negative side, we only give polynomial, and not strongly polynomial algorithms (exact or approximation) for design problems, whenever the optimization problem has a polynomial time (exact or approximation) algorithm.

Two prominent design problems studied recently are: Boyd, Diaconis and Xiao [12] study the design of the fastest mixing Markov chain on a graph with a budget constraint on the weights of the edges of a fixed graph. Elson, Karp, Papadimitriou and Shenker [33] study the synchronization design problem in sensornets, which is essentially the problem of finding a Markov chain on a graph that minimizes the maximum commute time.

5.1 Formal definition of design

We present a general framework to define the design versions of optimization problems:

Definition 4 *An **optimization problem** Π consists of a set of* valid instances $\mathcal{I}_\Pi$*, and an objective function obj. Each instance I is a tuple $(E_I, \mathcal{S}_I, \mathbf{w}_I)$. Henceforth we will drop*

the subscript when the instance is clear from context. E is a universe of elements*, and each element $e \in E$ has an associated weight $\mathbf{w}(e) \geq 0$, a rational number, giving the vector $\mathbf{w}$. Throughout we will let $n = |E|$. Each instance also has a set of* feasible solutions [1] *$\mathcal{S}$. For an instance $I = (E, \mathcal{S}, \mathbf{w})$, and a feasible solution $S \in \mathcal{S}$, the value of the objective function is $obj(I, S)$. We restrict this to be a function of S and $\mathbf{w}$ only, so we may write this as $f_S(\mathbf{w})$, for some function f_S. For a maximization problem, the goal is to find an optimal solution:*

$$S^* = \operatorname*{argmin}_{S \in \mathcal{S}} f_S(\mathbf{w})$$

We also define:

$$OPT_{\Pi}((E, \mathcal{S}, \mathbf{w})) = \min_{S \in \mathcal{S}} f_S(\mathbf{w})$$

For $\alpha \geq 1$, a feasible solution S' is called an α-approximate solution to I if:

$$f_{S'}(\mathbf{w}) \leq \alpha \cdot OPT_{\Pi}(I)$$

An algorithm is called an α-approximation algorithm for the problem Π if for every instance I of Π, the algorithm returns an α-approximate solution to I. The goal of a maximization problem is defined similarly.

Definition 5 *The **maxmin design version** $D(\Pi)$ of a minimization problem Π is defined as follows: For every collection of valid instances of Π of the form $I = (E_I, \mathcal{S}_I, \cdot)$, there is one valid instance of $D(\Pi)$: $J = (E_J, \mathcal{S}_J, B_J)$, where $E_J = E_I$, $\mathcal{S}_J = \mathcal{S}_I$, and B_J is a rational number, called the weight budget. A feasible solution to J is a weight vector $\mathbf{w} = (\mathbf{w}(e))_{\{e \in E_J\}}$, which satisfies the* budget constraint *$\sum_{e \in E_J} \mathbf{w}(e) \leq B_J$. Every feasible solution $\mathbf{w}$ to J leads to an instance $I = (E_I, \mathcal{S}_I, \mathbf{w})$ of the optimization problem Π.*

The goal of the maxmin design problem is to find a feasible solution $\mathbf{w}$ so that the minimum objective function value of the resulting instance of the minimization problem is as large as possible. That is, the goal is to find an optimal solution:

$$\mathbf{w}^* = \operatorname*{argmax}_{\mathbf{w}: \sum_e \mathbf{w}(e) \leq B} OPT_{\Pi}((E, \mathcal{S}, \mathbf{w}))$$

We also define:

$$OPT_{D(\Pi)}((E, \mathcal{S}, B)) = \max_{\mathbf{w}: \sum_e \mathbf{w}(e) \leq B} OPT_{\Pi}((E, \mathcal{S}, \mathbf{w}))$$

For $\alpha \geq 1$, a weight vector $\mathbf{w}'$ is called an α-approximate solution to I if:

$$OPT_{\Pi}((E, \mathcal{S}, \mathbf{w})) \geq \frac{1}{\alpha} \cdot OPT_{D(\Pi)}$$

An algorithm is called an α-approximation algorithm for a design problem $D(\Pi)$ if for every instance I of $D(\Pi)$, the algorithm returns an α-approximate solution to I.

The minmax design version of a maximization problem is defined similarly.

[1] *The number of feasible solutions may be exponential in $n = |E|$*

Definition 6 *An optimization problem* Π *(and its design version* $D(\Pi)$*) is called* ***linear*** *if all its instances* $I = (E, \mathcal{S}, \mathbf{w})$*, are of the following form:* $\mathcal{S} \subseteq 2^E$*, and* $\forall\ S \in \mathcal{S}:\ f_S(\mathbf{w}) = \sum_{e \in S} a_{e,S}\mathbf{w}_e$*, for some* $a_{e,S} \geq 0$*. A more general class of problems has the functions* f_S *being convex or concave functions of* $\mathbf{w}$*. We shall call these* ***concave minimization*** *and* ***convex maximization*** *problems.*

Examples: Most optimization problems on graphs are linear, as defined above. For example, in the Min-Steiner-Tree problem (Traveling-Salesman, Sparsest Cut), an instance $I = (E, \mathcal{S}, \mathbf{w})$ has E being the set of edges of the given graph, $\mathcal{S}$ being the collection of all sets of edges which form Steiner trees (Hamiltonian cycles, cuts), and $\mathbf{w}$ being the given weights on the edges. An instance $(E, \mathcal{S}, B)$ of the design version of these problems would be to allocate a budget of B to the edges of the graph so as to maximize the weight of the minimum weight Steiner tree (maximize the weight of the best TSP tour, make the sparsest cut as dense as possible). Clearly, some design problems make more intuitive sense than others.

An example of a convex maximization problem is that of finding the maximum commute time of a random walk on a graph over different pairs of vertices. Here an instance is $I = (E, \mathcal{S}, \mathbf{w})$, where E is the set of edges of the graph, $\mathcal{S}$ is the collection of pairs of vertices, and the w_es are the relative conductances of the edges, giving the transition probabilities. The functions f_S are the commute time functions, known to be convex. The design version of this problem is that of assigning transition probabilities to minimize the maximum commute time.

5.2 Solving design problems

5.2.1 A general technique based on the ellipsoid method

Consider a concave minimization problem and its corresponding max-min design problem. The analysis for convex min-max design problems is similar. The following theorem states that if the optimization version can be solved in polynomial time, then the design version can be solved up to additive error ϵ using the ellipsoid method.

Theorem 5.2.1 *If we have an algorithm which solves the minimization problem* $\Pi = (E, \mathcal{S}, \mathbf{w})$ *with a concave objective function* $f_S(\mathbf{w})$ *in polynomial time, then for any* $\epsilon > 0$*, we can solve the corresponding max-min design problem* $D(\Pi)$ *up to an additive error of* ϵ *in time polynomial in* n *and* $\log \frac{1}{\epsilon}$*.*

Proof: Note that the value of the optimal max-min design is given by the following program:

$$OPT_{D(\Pi)} := \max\{\lambda :\ \lambda - f_S(\mathbf{w}) \leq 0,\ \forall S \in \mathcal{S};\quad \sum_{e \in E} \mathbf{w}(e) \leq B;\quad \mathbf{w}(e) \geq 0,\ \forall e \in E\} \quad (20)$$

If $f_S()$ is concave, then the above program is also convex, since for any two feasible solutions $(\lambda, \mathbf{w})$ and $(\lambda', \mathbf{w}')$ and $0 \leq \mu \leq 1$, we have

$$f_S(\mu\mathbf{w} + (1-\mu)\mathbf{w}') \;\geq\; \mu f_S(\mathbf{w}) + (1-\mu) f_S(\mathbf{w}') \;\geq\; \mu\lambda + (1-\mu)\lambda'$$

Hence, one can solve the above program using the ellipsoid method. Assuming an upper bound F on the optimum, the algorithm proceeds with a guess λ of the optimum to the program and tests for emptiness of the convex feasible set. For any $\epsilon > 0$, in time polynomial in the input size and $\log \frac{1}{\epsilon}$, the ellipsoid method (see [46]) using the minimization problem as a separating oracle[2] to construct the ellipsoids, returns a feasible $\mathbf{w}$, or asserts that optimum is smaller than $\lambda + \epsilon$. Via a binary search to find λ^* which takes time $\log F$, the theorem follows by noting that an upper bound on λ^* is polynomial in the size of the value returned by the minimization problem. □

In the next theorem we show if the minimization problem has an α-approximation, then so does the max-min design version.

Theorem 5.2.2 *If we have a polynomial time algorithm returning an α-approximation to the optimization problem Π, then we can find, for any $\epsilon > 0$, an approximation algorithm for the design problem $D(\Pi)$, with a multiplicative factor of α and an additive error of ϵ.*

Proof: We have a polytime algorithm which, given $(E, \mathcal{S}, \mathbf{w})$, returns a set S with objective function value guaranteed to be at most α-factor away from the actual optimum: $f_S(\mathbf{w}) \leq \alpha \min_{T \in \mathcal{S}} f_T(\mathbf{w})$. As in the proof of Theorem 5.2.1, given a guess λ, we run ellipsoid to check if there exists a feasible $\mathbf{w}$. The difference now is that the separation oracle is the approximate minimization algorithm. Thus, for any $\epsilon > 0$, the ellipsoid algorithm returns, in time polynomial in input and $\log \frac{1}{\epsilon}$, a solution $(\lambda, \mathbf{w})$, so that the optimum is less than $\lambda + \epsilon$. However, since the separation oracle is approximate, $(\lambda, \mathbf{w})$ might not be feasible itself. Nevertheless, by the guarantee of the approximation, we know $(\lambda/\alpha, \mathbf{w})$ is feasible, which implies the theorem. □

Remark 5.2.3 *We note that for linear optimization problems where the function $f_S()$ is linear, the program 20 is a linear program, and (see [46]) the above two theorems hold without any additive error.*

The ellipsoid method may need to take a number of steps equal to a large polynomial. In each step we need to solve an instance of the optimization problem Π. The ellipsoid method also takes a huge time in practice. This motivates us to look for faster algorithms for the design problem. In Section 5.3, we will provide a different general method which works much faster.

[2]Here, and throughout, we will say that an (approximation) algorithm solves a optimization problem if it gives the (approximately) optimum value as well as a set S which achieves this (approximately) optimum value.

5.2.2 A technique based on LP-relaxation

In this section we describe a general technique for solving design problems, in the case that we have a linear programming relaxation for the minimization problem Π. This technique is similar to that of Jüttner [54] and will be used in the coming chapter to obtain results about packing.
Suppose we have:

$$OPT_{\Pi} \geq \min \{ \quad \mathbf{w} \cdot \mathbf{x} \quad \text{s.t} \quad A\mathbf{x} \geq \mathbf{b}; \quad \mathbf{x} \geq \mathbf{0} \quad \} \tag{21}$$

Moreover, suppose there is an α-approximate polynomial time algorithm which returns a solution S with $f_S(\mathbf{w}) \leq \alpha \cdot L \leq \alpha \cdot OPT_{\Pi}$, where L is the solution to LP(21) (That is, the integrality gap of the LP is at most α). Then we have an α-approximation for the design version as well.

Theorem 5.2.4 *If we have an LP relaxation for the optimization problem Π, and a polynomial time algorithm producing a solution within $\alpha \geq 1$ times the LP optimum, then we can produce an α approximation algorithm for the corresponding design problem $D(\Pi)$ which requires solving a single LP having one constraint more than that of the LP relaxation.*

Proof: Look at the dual of LP(21).

$$\max \{ \quad \mathbf{b} \cdot \mathbf{y} \quad \text{s.t} \quad \mathbf{y}^T A \leq \mathbf{w}; \quad \mathbf{y} \geq \mathbf{0} \quad \} \tag{22}$$

In the design problem, note that the weight vector $\mathbf{w}$ is no longer in the objective function but appears in the constraints. Parameterizing the program on $\mathbf{w}$, let the optimal solution to LP 22 be $D(\mathbf{w})$. From the previous supposition, we know there is an algorithm giving a set S with the guarantee, $D(\mathbf{w}) \leq f_S(\mathbf{w}) \leq \alpha D(\mathbf{w})$ for all weight vectors $\mathbf{w}$.

To solve the design problem, we consider $\mathbf{w}$ as a variable in LP 22, and add the constraint that the total weight is bounded by B. Thus we solve the following LP

$$\max \{ \quad \mathbf{b} \cdot \mathbf{y} \quad \text{s.t} \quad \mathbf{y}^T A - \mathbf{w} \leq \mathbf{0}; \quad \mathbf{w} \cdot \mathbf{1} \leq B; \quad \mathbf{y}, \mathbf{w} \geq \mathbf{0} \quad \} \tag{23}$$

Let the optimal solution to LP 23 be D^*. Let $\mathbf{w}'$ be the optimum vector returned in the solution of LP 23. Note that for any weight vector $\mathbf{w}$ satisfying $\mathbf{w} \cdot \mathbf{1} \leq B$, we have $D(\mathbf{w}) \leq D^*$ with equality at $\mathbf{w}'$. Solve LP 21 with $\mathbf{w}'$ and obtain a set T with the guarantee $D^* \leq f_T(\mathbf{w}') \leq \alpha D^*$.

We now claim that $T, \mathbf{w}'$ gives an α approximation to the design problem. To see this, suppose $\mathbf{w}^*$ was the weight vector achieving the maxmin design. Moreover, suppose S was the set that minimized its objective value given $\mathbf{w}^*$. We need to show $\alpha f_T(\mathbf{w}') \geq f_S(\mathbf{w}^*)$. To see this note $f_S(\mathbf{w}^*) \leq \alpha D(\mathbf{w}^*) \leq \alpha D^* \leq \alpha f_T(\mathbf{w}')$. □

As a corollary we get a $\log n$ approximation to maximum min-multicut, a 2-approximation to the maximum min weighted vertex cover, a 2-approximation for max-min Steiner trees and many such problems which have approximation algorithms via LP-relaxations.

5.3 Faster algorithms for Design Problems

In this section we use techniques from learning theory to obtain faster algorithms for design problems. We present the technique here for linear optimization problems. Assuming via scaling that $B = 1$, the general theorem states that given an α-approximation to the linear optimization problem, one can find an α-approximation to the design problem in time $O(\log n/\epsilon^2)$ with additive error ϵ.

5.3.1 Linear Design Problems, Zero-sum Games and Multiplicative Updates

Recall the definition of linear optimization problems and their design versions: the instances $I = (E, \mathcal{S}, \mathbf{w})$, are of the form $\mathcal{S} \subseteq 2^E$, and $\forall\ S \in \mathcal{S}:\ f_S(\mathbf{w}) = \sum_{e \in S} a_{e,S}\mathbf{w}_e$. In this section we shall take all the $a_{e,S} = 1$ for the sake of succinct notation – all the proofs extend naturally to the general case – so that $f_S(\mathbf{w}) = \sum_{e \in S} \mathbf{w}_e$. Moreover, we will assume $B = 1$ as this is only a scaling factor for linear optimization problems.

Definition 7 *Given an instance $I = (E, \mathcal{S}, B{=}1)$ of a maxmin design problem $D(\Pi)$, the* **equivalent zero-sum game** *$G(I)$ is defined by an $|E| \times |\mathcal{S}|$ matrix as follows: the rows are indexed by E and the columns by $\mathcal{S}$, and the entry $(e, S) = 1$ if $e \in S$, 0 otherwise. The entries of the matrix represent the payment of the column player to the row player.*

The game $G(I)$ is equivalent to the instance I of the design problem in the following way: A mixed strategy $\mathbf{x}$ of the row player in $G(I)$ corresponds to a weight distribution $\mathbf{w}$ in I. Given a mixed strategy $\mathbf{x}$ of the row player, the payment to the row player for the pure strategy S of the column player is precisely the value of the objective function $f_S(\mathbf{w})$ in I. Thus, given the row's mixed strategy $\mathbf{x}$, if the column player plays its best response to $\mathbf{x}$, then the payment to the row player is precisely $OPT_\Pi(E, \mathcal{S}, \mathbf{x})$. Finally, this means that the set of maxmin strategies of the row player in $G(I)$ is precisely the set of solutions to the instance I of the design problem $D(\Pi)$. The value of the game $G(I)$ is precisely $OPT_{D(\Pi)}(I)$.

The technique of multiplicative updates can be used to find approximate maxmin strategies of a zero-sum game much faster than by solving a linear program [38]. In this section we describe this technique in terms of solving instances of design problems. The algorithms and proofs here follow the proofs of [38] as applied to our setting. The multiplicative updates technique has proved to be extremely useful in a wide array of applications in computer science - see e.g., the recent survey paper by Arora et al. [5]. We show here how this technique can be used to transform an α-approximation algorithm for an optimization problem to an α-approximation algorithm for its design version (for every α). We describe the algorithm in Algorithm 14.

Intuitively, at each step t, the algorithm finds the minimum solution with respect to weights $\mathbf{w}_t$, and then in the next step increases the weights on the elements in the solution

Algorithm 14 DESIGN-LINEAR

- **Input:** Instance $I = (E, \mathcal{S}, B = 1)$.
- **Parameters:** Real $\beta > 1$, integer $T > 1$, to be fixed later.
- **Output:** Weight vector $\overline{\mathbf{w}}$, an α-approximate solution to I.

1. **Initialize** $\forall\ e : z_1(e) = 1$. Let $\mathbf{w}_1(e) = z_1(e)/\sum_e z_1(e)$.
2. **Multiplicative update:** For $t = 1, \ldots, T$, do:
   - Suppose $\mathcal{A}$ on input $\mathbf{w}_t$ returns solution S_t.
   - $z_{t+1}(e) = z_t(e)\beta^{\mathbf{1}_{(e,S_t)}}$, where $\mathbf{1}_{(e,S_t)} = 1$ if S_t contains e, 0 otherwise;
   - $\mathbf{w}_{t+1}(e) = z_{t+1}(e)/\sum_e z_{t+1}(e)$
3. Return $\overline{\mathbf{w}} := \frac{B}{T}\sum_{t=1}^{T} \mathbf{w}_t$

returned. To analyze the algorithm, following [38] we define the quantity *regret* as

$$R_T := \max_{\mathbf{w}:\sum_e \mathbf{w}(e)=1} \sum_{t=1}^{T} f_{S_t}(\mathbf{w}) - \sum_{t=1}^{T} f_{S_t}(\mathbf{w}_t)$$

The following theorem was proved in [38].
Theorem [38]: Fixing the choice of $\beta = 1 + \sqrt{\frac{2\ln n}{T}}$, gives us

$$R_T \leq \sqrt{T}O(\sqrt{\ln n})$$

Now we are ready to prove the bound on the quality of our solution.

Theorem 5.3.1 *For every $\epsilon > 0$, given an α-approximation algorithm $\mathcal{A}$ to a linear minimization problem Π, algorithm* DESIGN-LINEAR*, when run for $T = O(\frac{\log n}{\epsilon^2\alpha^2})$ rounds, returns an α-approximate solution to every instance I of the maxmin problem $D(\Pi)$, up to an additive error $\epsilon > 0$.*

Proof: We need to argue about the quantity $\min_S f_S(\overline{\mathbf{w}})$. In the following, when we use subscript $\mathbf{w}$ we assume that sum of weights is equal to 1, and that the weights will be scaled

to sum to B at the end.. We follow the proof as in [38]. We have

$$\begin{aligned}
\min_S f_S(\overline{\mathbf{w}}) &= \min_S \tfrac{1}{T}\sum_{t=1}^{T} f_S(\mathbf{w}_t) && \text{(by linearity of } f_S\text{)}\\
&\geq \tfrac{1}{T}\sum_{t=1}^{T} \min_S f_S(\mathbf{w}_t)\\
&\geq \tfrac{1}{T}\sum_{t=1}^{T} \tfrac{1}{\alpha} f_{S_t}(\mathbf{w}_t) && (\mathcal{A} \text{ is an } \alpha\text{-approximation algorithm})\\
&\geq \tfrac{1}{\alpha}\max_{\mathbf{w}} \tfrac{1}{T}\sum_{t=1}^{T} f_{S_t}(\mathbf{w}) - O(\tfrac{1}{\alpha}\sqrt{\tfrac{\ln n}{T}}) && \text{(by Theorem of Freund-Schapire)}\\
&\geq \tfrac{1}{\alpha}\max_{\mathbf{w}} \min_S f_S(\mathbf{w}) - O(\tfrac{1}{\alpha}\sqrt{\tfrac{\ln n}{T}}) && \text{(minimum is smaller than the average)}
\end{aligned}$$

□

A multiplicative-error version can also be derived as follows.

Corollary 5.3.2 *If the α-approximation algorithm $\mathcal{A}$ for Π runs in time $T_{\mathcal{A}}$, then given any $\epsilon > 0$, algorithm* DESIGN-LINEAR *is $(1+\epsilon)\alpha$-approximate and runs in time $O(T_{\mathcal{A}}\frac{n^2 \ln n}{\epsilon^2})$.*

Proof: The proof follows by noting that the optimum design solution must have weight more than $1/n$. The corollary follows by plugging in the value $\epsilon' = \epsilon/n$ in the previous theorem. □

5.4 The complexity of design problems

In this section we study the relationship of the complexity of design problems and the complexity of the corresponding optimization problems.

The main result of this paper as described in Sections 5.2 and 5.3 is that solving a design problem $D(\Pi)$ is as easy as solving the corresponding optimization problem Π, for the class of concave (convex) minimization (maximization) problems, up to arbitrarily small additive errors. This is proved in Theorem 5.2.2. For linear optimization problems, if Π is in **P** then $D(\Pi)$ is also in **P** (see Remark 5.2.3 in Section 5.2.1). This may not be true for convex or concave optimization problems, since it may be that Π is in **P**, but all optimal solutions for $D(\Pi)$ have irrational values. However, we can still solve $D(\Pi)$ upto an arbitrarily small additive approximation in polynomial time.

A natural question to ask is if the converse also holds, i.e. whether solving the optimization problem is as easy as the design version of the same. The following shows that this is not the case:

Theorem 5.4.1 *There exists an linear minimization problem Π such that finding the value of the minimum is* **NP***-complete, but its design version $D(\Pi)$ can be solved in polynomial time.*

Proof: Call a graph a bridged clique if it consists of two cliques K_1 and K_2, and two edges $(u, u'), (v, v')$ with $u, v \in K_1$ and $u', v' \in K_2$. Consider the problem of finding (the value of) the cheapest tour on a weighted bridged clique. This problem is **NP**-hard as it involves finding the cheapest hamiltonian paths between u, v and u', v' respectively. Now consider the design version of the problem. We have to find a distribution of the weight budget on a bridged clique so that the cost of the minimum weight tour is maximized. Since any tour will have to pick both edges of the bridge, the optimal strategy is to divide the weights only on the bridge edges. Thus the design version of this problem can be solved trivially in polynomial time. This construction extends to any **NP**-complete problem. □

We have seen that all design problems are as easy as their optimization versions (up to additive errors), and that some are polynomial time solvable even though the optimization versions are **NP**-hard. To complete the picture we show below that not all design problems are easy:

Theorem 5.4.2 *There exists an* **NP***-complete linear minimization problem such that the corresponding design problem is also* **NP***-complete.*

Proof: Consider the problem of finding the minimum weight Steiner tree in a weighted graph. We prove in Section 6.2 of Chapter 6 (Theorem 6.1.1) that the value of the maxmin Steiner tree is exactly the reciprocal of the maximum number of Steiner trees that can be fractionally packed in the weighted graph. However, the fractional packing number of Steiner trees is known to be **NP**-hard, as proved by Jain et al. [52]. □

5.5 Discussion: Design versions of Counting Problems

In this paper we gave a systematic way of going from an optimization problem to a design problem, and we studied their relative complexity. We leave open the issue of carrying out an analogous plan of study for counting problems, in particular, #P-complete problems.

Two rather interesting design problems that arise from counting problems are the following. Determining their complexity is by itself a challenging open problem.

1. **Network reliability:** Given probabilities of edge failures in an undirected graph, the problem of determining the probability that the graph gets disconnected is #P-complete [82]. An FPRAS (fully polynomial randomized approximation scheme) for this problem was given by Karger [55].

   Consider the following design version of this problem. Let $G = (V, E)$ be an undirected graph. With each edge e we are specified a number p_e such that $0 < p_e < 1$. We are given a total weight of W. If weight w is placed on edge e then its failure probability becomes p_e^w. The problem is to determine the optimal way of placing weight W on the edges so that the failure probability of the resulting graph is minimized.

2. **Permanent:** We will define a design version of the problem of computing the permanent of a non-negative matrix. Our problem turns out to be a generalization of the van der Waerden Conjecture, which was settled positively by Falikman [34] and Egorychev [32]. This conjecture states that the matrix that has all entries $1/n$ is the doubly stochastic $n \times n$ matrix that has minimum permanent.

   Let A be an $n \times n$ 0/1 matrix whose permanent is non-zero. This is the *template matrix*. The problem is to replace the entries that are 1's in A by non-negative numbers so that the permanent of the resulting matrix is the minimum possible subject to the condition that it is doubly stochastic.

CHAPTER VI

FRACTIONAL AND INTEGRAL PACKING OF TREES

Given a set system $(E, \mathcal{S} \subseteq 2^E)$, the *maximum unweighted disjoint set packing problem* asks what is the maximum cardinality subfamily of $\mathcal{S}$ such that the members of the subfamily are pairwise disjoint. Packing problems form a large part of combinatorial optimization, and in this chapter we investigate the *tree-packing problem*: where the element set E is the edge-set of an undirected graph and the family $\mathcal{S}$ are subsets of edges which induce trees.

Packing problems are closely related to design versions of problems (actually the max-min design version). Given an minimization problem instance $I = (E, \mathcal{S}, \mathbf{w})$, note that if in the set system $(E, \mathcal{S})$ one could pack k disjoint sets, the max-min design version of the minimization problem with $B = 1$ would be only smaller than $1/k$. In Section 6.1, we actually show that the optimum design equals the reciprocal of the *fractional packing* number of the set system.

The above result along with the main theorem of Chapter 5 implies that fractional packing is as easy as optimization. Moreover, in the special case of packing trees, we will use the framework developed in Section 5.2.2 of Chapter 5 to obtain some results about fractionally packing trees. Finally, in the last section of the chapter, we extend our results to integral packing (the disjoint packing problem stated above) of *spanning trees* in a graph obtaining an alternate proof of the around five-decade old theorem of Nash-Williams and Tutte [78, 91].

6.1 Maxmin design is equivalent to fractional packing

Consider the tuple $\mathcal{F} = (E, \mathcal{S})$ as a general set system. The fractional packing number $k_f(\mathcal{F})$ is defined as the maximum number of fractionally disjoints sets in $\mathcal{S}$, that is,

$$k_f(\mathcal{F}) := \max\{\sum_{S \in \mathcal{S}} \lambda_S : \sum_{S: e \in S} \lambda_S \leq 1, \ \forall e \in E; \quad \lambda_S \geq 0, \ \forall S \in \mathcal{S}\}$$

Theorem 6.1.1 *For any set system $\mathcal{F} = (E, \mathcal{S})$, we have $k_f(\mathcal{F}) = 1/OPT_{D(\Pi)}(E, \mathcal{S}, 1)$, that is, the fractional packing number equals the reciprocal of the maxmin design of the linear instance $(E, \mathcal{S})$ given budget of 1.*

Proof: By duality, we can write $k_f(\mathcal{F})$ as

$$k_f(\mathcal{F}) = \min\{\sum_{e \in E} x_e : \sum_{e \in S} x_e \geq 1, \ \forall S \in \mathcal{S}; \quad x_e \geq 0, \ \forall e \in E\} \tag{24}$$

By definition (LP(20)),

$$OPT_{D(\Pi)}(E,\mathcal{S},1) = \max\{\lambda : \sum_{e\in S} x_e \geq \lambda,\ \forall S \in \mathcal{S};\quad \sum_{e\in E} x_e = 1;\quad x_e \geq 0,\ \forall e \in E\} \quad (25)$$

We complete the proof by showing that the optimal solution of LP 24 is the reciprocal of optimal solution of LP 25. Take an optimal solution $\{x_e\}_{e\in E}$ to LP 24 of value k_f. Note that $(1/k_f, \{w_e = x_e/k_f\}_{e\in E})$ is a feasible solution for LP 25. This is because $\sum_{e\in E} w_e = \sum_{e\in E} x_e/k_f = 1$ and for all sets S, $\sum_{e\in S} w_e = \sum_{e\in S} x_e/k_f \geq 1/k_f$. Similarly, if $(\lambda, \{w_e\}_{e\in E})$ is a solution to LP 25, then $\{x_e = \frac{w_e}{\lambda}\}_{e\in E}$ is a solution to LP 24 of value $\frac{1}{\lambda}$. □

6.2 Fractionally packing Steiner trees

In this section and the next, we focus on the packing problem of trees in undirected graphs. In particular we study the problem of packing Steiner trees. We start with some preliminaries about Steiner trees, we refer the reader to Chapter 2 for more on Steiner trees.

Preliminaries: We work with a multigraph $G = (V, E)$. The vertex set is partitioned into two sets $V = R \cup S$. R is the set of required vertices, S the set of Steiner vertices. A Steiner tree is a minimal subgraph of G connecting all the vertices in R. We denote the set of Steiner trees of G as $\mathcal{T}(G)$, or simply $\mathcal{T}$ when the graph is clear from context. The *integral packing number* of G is the maximum number of edge-disjoint Steiner trees in $\mathcal{T}$ and is denoted by $k_{int}(G)$. The *fractional packing number* of G is the maximum number of Steiner trees in $\mathcal{T}$ that can be packed fractionally and is denoted by $k_f(G)$. Thus,

$$k_{int}(G) := \max\{\sum_{T\in\mathcal{T}} \lambda(T) : \sum_{T:e\in T} \lambda(T) \leq 1, \forall e \in E;\quad \lambda(T) \in \{0,1\}, \forall T \in \mathcal{T}\} \quad (26)$$

$$k_f(G) := \max\{\sum_{T\in\mathcal{T}} \lambda(T) : \sum_{T:e\in T} \lambda(T) \leq 1, \forall e \in E;\quad \lambda(T) \geq 0, \forall T \in \mathcal{T}\} \quad (27)$$

Let Π, the Steiner tree polytope, be the convex hull of indicator vectors of the Steiner trees of G, that is, $\Pi := \{x \in \mathbb{R}^E : x = \mathbf{1}_T, T \in \mathcal{T}\}$. $\mathbf{1}_T$ is the vector in $\mathbb{R}^E$ with a 1 corresponding to edges in T and 0 otherwise. Given weights $w(e)$ on each edge of G, the minimum weight Steiner tree would be denoted as $MST(w)$. Note that $MST(w) = \min\{\sum_e w(e)\cdot x(e) : x \in \Pi\}$. Determining $MST(w)$ in general graphs is NP-hard. $\Pi' \subseteq \mathbb{R}^E$ is a relaxation if $\Pi \subseteq \Pi'$. The LP $\min\{\sum_e w(e)\cdot x(e) : x \in \Pi'\}$ is called an LP relaxation for the Steiner tree problem and is denoted as $L_{\Pi'}(w)$.

The *max-min Steiner tree* problem is to find a weight assignment $w : E \to \mathbb{R}^+$, so that $\sum_{e\in E} w(e) = 1$ and the weight of the minimum weight Steiner tree is maximized. The

weight of the max-min Steiner tree is denoted as $MMST$. Thus,

$$MMST := \max\{MST(w) : \sum_{e\in E} w(e) = 1\}$$
$$= \max\{\nu : w(T) \geq \nu,\ \forall T \in \mathcal{T};\ \sum_{e\in E} w(e) = 1\}$$

The following observation is a special case of Theorem 6.1.1.

Theorem 6.2.1 *For any graph G, $MMST(G) = \frac{1}{k_f(G)}$*

A partition $\mathcal{P} = (V_1, V_2, \cdots, V_p)$ of the vertices V is called a *valid partition* if $p > 1$ and for all $i = 1 \cdots p$, $V_i \cap R \neq \varnothing$. The edges with each end point in separate partitions are called cross-edges of the partition, $E(\mathcal{P})$. The strength of a partition is the following ratio: $\frac{|E(\mathcal{P})|}{p-1}$, The Steiner strength of the graph, denoted by $\gamma(G)$ is the minimum strength over all valid partitions.

$$\gamma(G) := \min_{valid\mathcal{P}} \frac{|E(\mathcal{P})|}{|\mathcal{P}| - 1} \tag{28}$$

Related Work: In 1961, in two separate papers (in the same journal!), Nash-Williams[78] and Tutte[91], showed that in the case when there are no Steiner vertices, $k_{int}(G) = \lfloor \gamma(G) \rfloor$. They observed that this implied if a graph is $2k$-edge-connected, then there are k-edge-disjoint spanning trees. In 1999, Kriesell[64] conjectured the same when there are Steiner vertices, with the connectivity requirements restricted to the required vertices. Kriesell[65] proved this conjecture for the case when the degree of Steiner vertices is even using Mader's splitting theorem[74]. Frank et. al.[37], using generalizations of theorems of Nash-Williams and Tutte to regular hypergraphs, proved that in graphs with no Steiner-Steiner edges (quasi-bipartite graphs), Kriesell's conjecture is true with the 2 replaced by 3. For general graphs, Jain et.al.[52] proved Kriesell's conjecture with 2 replaced by (ignoring lower order terms) $|R|/4$. This was recently greatly improved to 24 in the remarkable work of Lap Chi Lau[67].

6.3 Results on fractional packing number via LP relaxations for minimum Steiner tree problem

Note that the max-min Steiner tree can be thought as

$$MMST = \max_{w:|w|=1} \min_{x\in\Pi} w \cdot x$$

where $|w|$ is the simply the sum $\sum_e w(e)$. To convert this into a pure maximization linear program, we take the dual of the program $L_\Pi(w) := \min_{x\in\Pi}\ w \cdot x$. Denote the dual as $D_\Pi(w)$. Thus, the following is an exact linear program for MMST

$$MMST := \max_{w:|w|=1} D_\Pi(w)$$

Obviously one doesn't expect an explicit characterization of $D_\Pi(w)$ given the NP-hardness of finding $MST(w)$. The observation now is that *every* LP relaxation for the minimum Steiner tree problem gives an LP relaxation for the max-min Steiner tree problem.

Let Π' be a relaxation of the Steiner tree polytope. Moreover suppose $\Pi' := \{x : Ax \geq \mathbf{1}; x \geq 0\}$ can be explicitly characterized[1]. $L_{\Pi'}(w) = \min\{w \cdot x : Ax \geq \mathbf{1}; x \geq 0\}$ is a lower bound on $MST(w)$. Taking the dual of $L_{\Pi'}(w)$, we get $D_{\Pi'}(w) = \max\{y \cdot \mathbf{1} : y^T A \leq w; y \geq 0\}$ is also a lower bound on $MST(w)$. This gives the following LP relaxation for $MMST$:

$$MMST \geq \max\{y \cdot \mathbf{1} : y^T A - w \leq 0; \quad |w| = 1; \quad y, w \geq 0\} \tag{29}$$

Moreover, if the integrality gap of the relaxation Π' is $\alpha(\Pi')$, that is, for any weight w, we have $L_{\Pi'}(w) \leq MST(w) \leq \alpha(\Pi') \cdot L_{\Pi'}(w)$, we get the integrality gap of the above relaxation is at most $\alpha(\Pi')$. That is This gives us:

$$MMST \leq \alpha(\Pi') \cdot \max\{y \cdot \mathbf{1} : y^T A - w \leq 0; \quad |w| = 1; \quad y, w \geq 0\} \tag{30}$$

Taking duals of the above LP, we get

$$MMST \leq \alpha(\Pi') \cdot \min\{\mu : Ax \geq 1; \ \mu \cdot \mathbf{1} - x \geq 0; \quad \mu, x \geq 0\} \tag{31}$$

In the remainder of the section we will plug in three different LP relaxations for the minimum Steiner tree problem and use the above framework and Theorem 6.2.1 to obtain three results about the fractional packing number.

The Undirected Cut Relaxation

Let $\mathcal{U} := \{U \subsetneq V : U \cap R \neq \varnothing \text{ and } U^c \cap R \neq \varnothing\}$ denote the subsets of V which contain at least one required vertex but not all. The undirected cut relaxation polytope Π_{UC} is the following.

$$\Pi_{UC} := \{x \in \mathbb{R}^E : x(\delta(U)) \geq 1, \ \forall U \in \mathcal{U}; \ x \geq 0\}$$

Plugging this in the inequality(31), we get

$$MMST \leq \ \alpha(\Pi_{UC}) \cdot \min\{\mu : x(\delta(U)) \geq 1, \forall U \in \mathcal{U}; \quad \mu \geq x(e), \forall e \in E; \ \mu, x \geq 0\}$$

This immediately implies the fractional version of Kriesell's conjecture. The only other paper to explicitly mention this is the work of Li and Li[72] who prove it using Mader's splitting theorem. The only tool we use is LP-duality.

Theorem 6.3.1 *If between any two required vertices there are $2k$ edge-disjoint paths, then $k_f(G) \geq k$.*

[1]More generally, Π' could be $\{x : [A \mid B](x \ t)^T \geq \mathbf{1}; x, t \geq 0\}$ where t are auxiliary variables. We omit this notation for the time being for brevity, although we will use auxiliary variables in the bi-directed cut relaxation below.

Proof: By Theorem (6.2.1), we need to prove $MMST \leq \frac{1}{k}$. Since it is known $\alpha(\Pi_{UC}) \leq 2$, it is enough to show a feasible solution of $\frac{1}{2k}$ for the above LP. Since between any two required vertices there are $2k$ edge-disjoint paths, by Menger's theorem $|\delta(U)| \geq 2k$ for all $U \in \mathcal{U}$ and thus $\mu = x(e) = \frac{1}{2k}$ for all edges e is a feasible solution. □

Multiway Cut Relaxation

The multiway-cut relaxation polytope Π_{MC} is the following.

$$\Pi_{UC} := \{x \in \mathbb{R}^E : x(E(\mathcal{P})) \geq |\mathcal{P}| - 1,\ \forall \text{ valid } \mathcal{P};\ x \geq 0\}$$

Plugging this in the inequality(31), we get

$$MMST \leq\ \alpha(\Pi_{MC}) \cdot \min\{\mu : x(E(\mathcal{P})) \geq |\mathcal{P}| - 1, \forall \text{valid } \mathcal{P};\quad \mu \geq x(e), \forall e \in E;\ \mu, x \geq 0\}$$

This leads us to the relation between fractional packing number and the Steiner strength of the graph.

Theorem 6.3.2 *For any graph, $k_f(G) \leq \gamma(G) \leq \alpha(\Pi_{MC}) \cdot k_f(G)$*

Proof: The first inequality follows from the definition of $k_f(G)$ and $\gamma(G)$. The second inequality follows from the solution $\mu = x(e) = \frac{1}{\gamma(G)}$ for all e, for the above LP. The feasibility of the above solution follows from definition of $\gamma(G)$. □

Corollary 6.3.3 *If the graph G has no Steiner vertices, that is, the Steiner trees are spanning trees, then $k_f(G) = \gamma(G)$.*

Proof: This follows from the fact that $\alpha(\Pi_{MC}) = 1$ for graphs with no Steiner vertices [25, 40]. □

Bi-directed cut relaxation

Call an arbitrary vertex $r \in R$ as the root. Bi-direct every edge of the multigraph and call the resulting set of arcs A giving each the same weight as the undirected edge. Let $\mathcal{U} := \{U \subsetneq V : U \cap R \neq \varnothing \text{ and } r \notin U\}$ denote the subsets of V which contain at least one required vertex but not the root. Consider the following polytope.

$$\Pi_{BC} := \{x \in \mathbb{R}^A : x(\delta^+(U)) \geq 1,\ \forall U \in \mathcal{U};\ x \geq 0\}$$

The bi-directed polytope is obtained by projecting onto $\mathbb{R}^E$ where the coordinate of an edge is the addition of the values on its two bi-directed arcs. We abuse notation and call the earlier polytope Π_{BC}. Note that, as was mentioned in a footnote before, Π_{BC} contains auxiliary variables: the arc variables. Thus, Π_{BC} cannot be plugged in directly into inequality(31), but rather into a more general inequality which can be derived similarly with the auxiliary variables. We omit the exposition of the calculation. This gives

$$MMST \leq\ \alpha(\Pi_{BC}) \cdot \min\{\mu : x(\delta^+(U)) \geq 1,\ \forall U \in \mathcal{U};\quad \mu \geq x(e_1) + x(e_2), \forall e \in E;\quad \mu, x \geq 0\} \tag{32}$$

where e_1 and e_2 are the bi-directed arcs of the undirected edge e.

We now digress a little to understand the forthcoming result. Consider the undirected multi-graph G as a communication network and root r as a source wishing to transmit to all the other nodes in R. Normally, such a communication is established via sending information on *multi-cast trees* which are nothing but Steiner trees. Edges have capacities (which can be modeled via multiplicity) and thus the maximum transmission rate is precisely the fractional packing number of the graph: Send $\lambda(T)$ amount of information across multicast tree T, where $\lambda(T)$ is the solution in the definition (26).

Over the past decade however, using a technique called network coding, the throughput has been increased. In fact the network coding principle [2] is the following: The maximum throughput achievable in a directed communication network between one source and many sinks is the minimum over all required vertices, the throughput between the source and that required vertex individually. Li and Li [72] noted that in an undirected graph this amounts to finding the optimum distribution of an undirected edge's capacity onto its bi-directed arcs so as to maximize the minimum throughput between the source and any required vertex. This is given via the following linear program whose optimum was called the *network coding throughput* and denoted as $\chi(G)$ [2].

$$\chi(G) := \max\{f : x(\delta^+ U) \geq f, \forall U \in \mathcal{U}; \quad x(e_1) + x(e_2) \leq 1, \forall e \in E; \quad x, f \geq 0\} \tag{33}$$

Comparing the definition of $\chi(G)$ and the inequality (32) obtained on plugging the bi-directed cut relaxation, the following theorem of Agarwal and Charikar [1] on the network coding gain ($\frac{\chi(G)}{k_f(G)}$) is immediate.

Theorem 6.3.4 *[1] For any undirected graph G, $k_f(G) \leq \chi(G) \leq \alpha(\Pi_{BC}) \cdot k_f(G)$.*

6.4 Integral packing of spanning trees

For this section assume that $S = \varnothing$. That is, we look at spanning trees. We prove the following theorem of Nash-Williams [78] and Tutte [91]:

Theorem 6.4.1 *If $S = \varnothing$, then $k_{int}(G) = \lfloor \gamma(G) \rfloor$.*

The proof has two steps. Call a graph G *critical* if it has an edge removing which decreases the integral packing number. The first step is to show that for critical graphs the integral packing number equals the strength. The second step is to use this property to obtain the desired result.

Definition 8 *A graph G is* critical *if there exists an edge $e \in E$ such that $k_{int}(G \setminus e) = k_{int}(G) - 1$. We call e the critical edge.*

[2] One can also think of this as a fractional version of the orientation problem in directed graphs

Lemma 6.4.2 *For critical graphs G with critical edge e, we have $k_{int}(G) = \gamma(G)$ and $\gamma(G \setminus e) < \gamma(G)$.*

Proof of Theorem(6.4.1)**:** The easy part is $k_{int}(G) \leq \gamma(G)$. To prove the equality, it is enough to show $\gamma(G) < k_{int}(G)+1$. Assume that there is at least one spanning tree T in G (that is G is connected). Let $T = \{e_1, e_2, \cdots, e_r\}$. Construct the following series of graphs $G =: G_0, G_1, \cdots, G_r$, where G_i is constructed by adding a copy of edge e_i to G_{i-1}. Note that $k_{int}(G_r) \geq k_{int}(G)+1$ since G_r has the extra edge-disjoint copy of T. Moreover, since we are adding one extra edge at each step, $k_{int}(G_i) \leq k_{int}(G_{i-1})+1$. Thus there exists some first G_i so that $k_{int}(G_i) = k_{int}(G_{i-1})+1 = k_{int}(G)+1$ implying G_i is critical. By Lemma(6.4.2), $\gamma(G) < \gamma(G_i) = k_{int}(G_i) = k_{int}(G)+1$ proving the theorem. □

Note that the above proof doesn't use the fact that $S = \varnothing$.

Proof of Lemma(6.4.2)**:** Let $k_{int}(G) = k$, and let e^* be the critical edge. Given a set of k disjoint trees, which we describe in a moment how we choose, we run a process described below which achieves the following goal: At each step, the process either (a) gives a partition of the $\mathcal{P}$ with $\frac{|E(\mathcal{P})|}{|\mathcal{P}|-1} = k$, or (b) increases the size of the partition, or (c) terminates. Moreover, the partition $\mathcal{P}$ satisfying (a) will contain the edge e^*, and thus $\gamma(G \setminus e^*) < \gamma(G)$. Since the size of a partition is smaller than the number of vertices, step (b) cannot happen forever, and thus either we prove our theorem or the process terminates. Our choice of disjoint trees is such that among all set of k disjoint trees, it terminates the earliest. We then show for such a set if the process terminates, then one can find k edge disjoint trees neither of which contain e^* contradicting criticality. It remains to describe the process.

Let $(T_1, T_2, \cdots, T_k)$ be a collection of edge disjoint spanning trees of G. Without loss of generality assume $e^* \in T_1$. Call the set of edges not in any of the trees as $N := E \setminus \cup_i T_i$.

Let V_1 and V_2 be the partition of vertices defined by T_1 and e^*. That is, V_1 and V_2 are the vertices of the two connected components formed when e^* is deleted from T_1. Call this partition $\mathcal{P}_1$ and initialize X_1 to be $E(\mathcal{P}_1)$. Note that $e^* \in E(\mathcal{P}_1)$. At each subsequent step, we will refine the partition $\mathcal{P}_i$ to get $\mathcal{P}_{i+1}$ and $X_{i+1} = E(\mathcal{P}_{i+1})$.

1. At step i, for all edges $e \in X_i$ let $w(e) := \frac{1}{r_i}$, where $r_i = |\mathcal{P}_i|$. Note that this weighing is feasible. If $|w| = k$, we are done. If $N \cap X_i \neq \varnothing$, we terminate the process. If neither happens, there must be a tree T_j with $|T_j \cap X_i| \geq (r_i + 1)$. There might be many, pick one arbitrarily and call it responsible for step $i+1$.

2. Since T_j uses at least $r_i + 1$ edges of $E(\mathcal{P}_i)$, there must be one component of $\mathcal{P}_i$, say V_r, so that V_r is disconnected in T_j. Form the partition $\mathcal{P}_{i+1}$ by refining V_r to be the various connected components of V_r in T_j.

Let $(T_1, T_2, \cdots, T_k)$ be such that the process terminates the earliest. If the process terminates at step 1, then it means there is an edge in $n \in N$ from V_1 to V_2. Define $T_1' = T_1 - e^* + n$ and note that $(T_1', T_2, \cdots, T_k)$ are k disjoint trees, none of which contains e^* contradicting the criticality of e^*.

Now we show that if the process terminates at step $t > 1$, then there is a set of k disjoint trees on which the process terminates earlier which will contradict the choice of the disjoint trees and complete the proof. By definition of termination, there exists a $n \in N$ which is in X_t but not in X_{t-1}. Suppose T_j is responsible for step t. By construction, $T_j + n$ contains a cycle whose edges intersect X_{t-1}. Let e be such an edge. Thus, $T_j' := T_j + n - e$ is a spanning tree. Consider the set of disjoint trees: $(T_1, \cdots, T_j', \cdots, T_k)$. Since the edge $e \in N'$ where N' is the new set of edges not in any tree, note that the process on these k disjoint trees will now terminate at step $t - 1$. □

We end with the following corollary which relates the fractional packing number and the integral packing number of trees in graphs with no Steiner vertices.

Corollary 6.4.3 *If $S = \varnothing$, then $k_{int}(G) = \lfloor k_f(G) \rfloor$.*

Proof: Follows from Corollary 6.3.3 and Theorem 6.4.1. □

6.5 Discussion

In Section 6.2, we saw that the plugging in of various LP relaxations of the minimum Steiner tree into the general framework of max-min Steiner trees (inequalities (29),(30) and (31)) give non-trivial results about the fractional packing number. Such a study is by no means restricted to the case of Steiner trees but holds for any optimization problem and might give interesting results for the case in hand. To give an example, the following fractional version of Lau's conjecture [66] on packing subgraphs with specified connectivity requirements can be proved by similar means mentioned in Section 6.2:

Theorem 6.5.1 *Given an undirected multigraph G and connectivity requirements r_{uv} for each pair of vertices (u, v), if there are $2k \cdot r_{uv}$ edge-disjont paths between u and v, then the fractional packing number for subgraphs satisfying the connectivity requirements is at least k.*

In Section 6.4, we have seen how critical instances of problems could possibly be used to prove strong relations between fractional and integral packing numbers (Corollary 6.4.3). In particular, we re-proved the classical theorem of Nash-Williams and Tutte about packing spanning trees going via critical graphs. We believe our proof of equality of the two numbers in these classes of graphs is new.

Our proof technique above can be extended to the problem of packing bases in a matroid. In particular, call a matroid critical if deleting an element from its ground set strictly

decreases the maximum number of disjoint bases that can be packed. We can prove the following theorem:

Theorem 6.5.2 *For a critical matroid $\mathcal{M}$, we have $k_f(\mathcal{M}) = k_{int}(\mathcal{M})$. Moreover, for any matroid $\mathcal{M}$, $k_{int}(\mathcal{M}) = \lfloor k_f(\mathcal{M}) \rfloor$*

We end this chapter with a conjecture that Corollary 6.4.3 extends for the general Steiner case.

Conjecture 6.5.3 *For any graph G, the maximum number of edge-disjoint Steiner trees that can be packed fractionally is within 1 of the maximum number of edge-disjoint Steiner trees that can be packed integrally.*

If our conjecture is true, then along with Theorem 6.3.1 this will prove Kriesell's conjecture upto an additive 1. We believe the method outlined above could be useful in proving so: that is, devising a notion of criticality and proving $k_f(G) = k_{int}(G)$ for critical graphs. However, we mention that there are examples which show that a similar generalization of the relation between integral packing number and strength of a graph (Theorem 6.4.1) is not true for general graphs.

REFERENCES

[1] Agarwal, A. and Charikar, M., "On the advantage of network coding for improving network throughput," in *Proceedings of the IEEE Information Theory Workshop*, pp. 2410 – 2424, 2004.

[2] Ahlswede, R., Cai, N., Li, S.-Y. R., and Yeung, R., "Network information flow," *IEEE Transactions on Information Theory*, vol. IT-46, pp. 1204–1216, 2000.

[3] Andelman, N., *Online and strategic aspects of network resource management algorithms*. PhD thesis, Tel Aviv University, Tel Aviv, Israel, 2006.

[4] Andelman, N. and Mansour, Y., "Auctions with budget constraints," *Proceedings of 9th Scandinavian Workshop on Algorithm Theory SWAT*, pp. 26–38, 2004.

[5] Arora, S., Hazan, E., and Kale, S., "The multiplicative weights update method: a meta algorithm and applications.," *Unpublished Manuscript*, 2006.

[6] Arora, S., Lund, C., Motwani, R., Sudan, M., and Szegedy, M., "Proof verification and the hardness of approximation problems.," *Journal of the ACM*, vol. 45, pp. 501–555, 1998.

[7] Azar, Y., Birnbaum, B., Karlin, A., Mathieu, C., and Nguyem, C., "Improved approximation algorithms for budgeted allocations," *To appear in Proceedings of* 35^{th} *International Colloquium on Automata, Languages and Programming (ICALP)*, 2008.

[8] Bansal, N., Khandekar, R., and Nagarajan, V., "Additive guarantees for degree bounded directed network design," *To appear in Proceedings of* 40^{th} *Annual ACM Symposium on the Theory of Computing (STOC)*, 2008.

[9] Benoit, J.-P. and Krishna, V., "Multiple object auctions with budget constrained bidders," *Review of Economic Studies*, vol. 68, pp. 155–179, 2001.

[10] Blumrosen, L. and Nisan, N., "Combinatorial auctions," in *Algorithmic Game Theory* (Nisan, N., Roughgarden, T., Tardos, E., and Vazirani, V. V., eds.), pp. 267–299, Cambridge University Press, 2007.

[11] Borgs, C., Chayes, J., Etesami, O., Immorlica, N., Jain, K., and Mahdian, M., "Bid optimization in online advertisement auctions," in *Proceedings of the 16th International Conference on World Wide Web*, 2007.

[12] Boyd, S., Diaconis, P., and Xiao., L., "The fastest mixing markov chain on a graph.," *SIAM Review*, vol. 46, no. 4, pp. 667–689, 2004.

[13] Buchbinder, N., Jain, K., and Naor, S., "Online primal-dual algorithms for maximizing ad-auctions revenue.," *Proceedings of 15th Annual European Symposium ESA*, pp. 253–264, 2007.

[14] Buchbinder, N. and Naor, J., "Online primal-dual algorithms for covering and packing problems.," in *13th Annual European Symposium on Algorithms (ESA)*, pp. 689–701, 2005.

[15] Buchbinder, N. and Naor, J., "Improved bounds for online routing and packing via a primal-dual approach.," in *Proceedings of* 47^{th} *IEEE Symposium on Foundations of Computer Science (FOCS)*, pp. 293–304, 2006.

[16] Carr, R. and Vempala, S., "Randomized metarounding," *Random Struct. and Algorithms*, vol. 20, pp. 343–352, 2002.

[17] Chakrabarty, D., Devanur, N. R., and Vazirani, V. V., "New geometry-inspired relaxations and algorithms for the Metric Steiner Tree Problem," *To appear in Proceedings of Integer Programming and Combinatorial Optimization IPCO*, 2008.

[18] Chakrabarty, D. and Goel, G., "On the Approxamibility of Budgeted Allocations and Improved Lower Bounds for Submodular Welfare Maximization and GAP," *Unpublished Manuscript*, 2008.

[19] Chakrabarty, D., Mehta, A., and Vazirani, V., "Design is as easy as optimization," *Proceedings of* 33^{rd} *International Colloquium on Automata, Languages and Programming (ICALP)*, pp. 477–488, 2006.

[20] Chakrabarty, D., Zhou, Y., and Lukose, R., "Budget constrained bidding in keyword auctions and online knapsack problems," *Appeared in 3rd Sponsored Search Workshop held in conjunction with WWW 2007, Banff, Alberta, Canada*, 2007.

[21] Chekuri, C. and Khanna, S., "A PTAS for the multiple-knapsack problem," *Proceedings of* 10^{th} *Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pp. 213–222, 2000.

[22] Chen, N., Engelberg, R., Nguyen, C., Raghavendra, P., Rudra, A., and Singh, G., "Improved approximation algorithms for the spanning star forest problem," *Proceedings of Approximation Algorithms for Combinatorial Optimization, International Workshop (APPROX)*, pp. 44–58, 2007.

[23] Chlebik, M. and Chlebikova, J., "Approximation hardness of the Steiner tree problem on graphs.," *Proceedings of Scandinavian Workshop on Algorithm Theory*, pp. 170–179, 2002.

[24] Chlebik, M. and Chlebikova, J., "Approximation hardness for small occurrence instances of NP-hard problems.," *Algorithms and Complexity, 5th Italian Conference, CIAC*, pp. 152–164, 2003.

[25] Chopra, S., "On the spanning tree polyhedron," *Operations Research Letters*, vol. 8, pp. 35–47, 1989.

[26] Chopra, S. and Rao, M. R., "The Steiner tree problem I: Formulations, compositions and extension of facets," *Math. Programming*, vol. 64, pp. 209–229, 1994.

[27] Chopra, S. and Rao, M. R., "The Steiner tree problem II: Properties and classes of facets," *Math. Programming*, vol. 64, pp. 231–246, 1994.

[28] Cormen, T., Leiserson, C., Rivest, R., and Stein, C., *Introduction to Algorithms, Second Edition*. MIT Press and McGraw-Hill, 2001.

[29] Courant, R., Robbins, H., and Stewart, I., *What Is Mathematics?: An Elementary Approach to Ideas and Method*.

[30] Dinur, I. and Safra, S., "The importance of being biased," *Proceedings of* 34^{th} *Annual ACM Symposium on the Theory of Computing (STOC)*, pp. 33–42, 2002.

[31] Edmonds, J., "Optimum branchings," *Journal of Research of the National Bureau of Standards. Section B*, vol. 71, pp. 233–240, 1967.

[32] Egorychev, G. P., "The solution of van der waerden's problem for permanents.," *Adv. Math*, vol. 42, pp. 299–305, 1981.

[33] Elson, J., Karp, R., Papadimitriou, C., and Shenker, S., "Global synchronization in sensornets.," *Proceedings of 6th Latin American Symposium (LATIN)*, pp. 609 – 624, 2004.

[34] Falikman, D. I., "Proof of the van der Waerden conjecture regarding the permanent of a doubly stochastic matrix.," *Mat. Zametki*, vol. 29, pp. 931–938, 1981.

[35] Feige, U. and Vondrák, J., "Approximation algorithms for allocation problems: Improving the factor of 1 - 1/e," *Proceedings of* 47^{th} *IEEE Symposium on Foundations of Computer Science (FOCS)*, pp. 667–676, 2006.

[36] Fleischer, L., Goemans, M. X., Mirrokini, V., and Sviridenko, M., "Tight approximation algorithms for maximum general assignment problems," *Proceedings of* 16^{th} *Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pp. 611–620, 2006.

[37] Frank, A., Kiraly, T., and Kriesell, M., "On decomposing a hypergraph into k connected sub-hypergraphs," *Discrete Applied Mathematics*, vol. 131, no. 2, pp. 373–383, 2005.

[38] Freund, Y. and Schapire, R., "Game theory, on-line prediction and boosting," *Proceedings of the* 9^{th} *Annual Conference on Computational Learning Theory (COLT)*, pp. 325–332, 1996.

[39] Fulkerson, D. R., "Increasing the capacity of a network: The parametric budget problem," *Management Science*, vol. 5(4), pp. 472–483, 1959.

[40] Fulkerson, D. R., "Blocking and anti-blocking pair of polyhedra," *Math. Programming*, vol. 1, pp. 168–194, 1971.

[41] Garg, R., Kumar, V., and Pandit, V., "Approximation algorithms for budget-constrained auctions," *Proceedings of Approximation Algorithms for Combinatorial Optimization, International Workshop (APPROX)*, pp. 102–113, 2001.

[42] Goemans, M., "Exercise 22.10 in [93]," 1996.

[43] Goemans, M. and Myung, Y., "A catalog of Steiner tree formulations," *NETWORKS*, vol. 23, pp. 19–23, 1993.

[44] Goemans, M. X. and Williamson, D. P., "Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming.," *Journal of the ACM*, pp. 1115–1145, 1995.

[45] Goemans, M. X. and Williamson, D. P., "The primal-dual method for approximation algorithms and its application to network design problems," in *Approximation Algorithms for NP-hard Problems* (Hochbaum, D. S., ed.), pp. 144–186, PWS Publishing Company, 1997.

[46] Grötschel, M., Lovász, L., and Schrijver, A., *Geometric Algorithms and Combinatorial Optimization*. Springer-Verlag, Berlin Heidelberg, 1988.

[47] Hästad, J., "Some optimal inapproximability results," *Journal of the ACM*, vol. 48, pp. 798–859, 2001.

[48] Hwang, F. K., Richards, D. S., and Winter, P., *The Steiner Tree Problem*, vol. 53 of *Annals of Discrete Mathematics*. Amsterdam, Netherlands: North-Holland, 1992.

[49] Ivanov, A. O. and Tuzhilin, A. A., *The Steiner problem and its generalizations*. BocaRaton, Ann Arbor, London, Tokyo: CRC Press, 1994.

[50] Iwama, K. and Taketomi, S., "Removable online knapsack problems," in 29^{th} *International Colloquium on Automata, Languages and Programming (ICALP)*, pp. 293–305, 2002.

[51] Jain, K., "A factor 2 approximation algorithm for the generalized Steiner network problem," *Combinatorica*, vol. 21, no. 1, pp. 39–60, 2001.

[52] Jain, K., Mahdian, M., and Salavatipour, M., "Packing Steiner trees," *Proceedings of* 13^{th} *Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pp. 266–274, 2003.

[53] Jain, K. and Vazirani, V. V., "Equitable cost allocations via primal-dual-type algorithms," in *Proceedings of 33rd ACM Symposium on Theory of Computing*, pp. 313–321, 2002.

[54] Jüttner, A., "On budgeted optimization problems.," *SIAM Journal on Discrete Mathematics*, vol. 20, pp. 880–892, 2006.

[55] Karger, D., "A randomized fully polynomial time approximation scheme for the all-terminal network reliability problem.," *SIAM Journal on Computing*, vol. 29, pp. 492–514, 1999.

[56] Karmarkar, N., "A new polynomial-time algorithm for linear programming.," *Combinatorica*, vol. 4, pp. 373–396, 1984.

[57] Karp, R., "Reducibility among combinatorial problems," *Complexity of Computer Computations*, pp. 85–103, 1972.

[58] Khachiyan, L., "A polynomial algorithm in linear programming," *Akademiia Nauk SSSR. Doklady*, vol. 244, pp. 1093–1096, 1979.

[59] Khot, S., "On the power of unique 2-prover 1-round games.," *Proceedings of* 34^{th} *Annual ACM Symposium on the Theory of Computing (STOC)*, pp. 767–775, 2002.

[60] Khot, S., Lipton, R., Markakis, E., and Mehta, A., "Inapproximability results for combinatorial auctions with submodular utility functions," *Proceedings of WINE*, pp. 92–101, 2005.

[61] Khot, S. and Regev, O., "Vertex cover might be hard to approximate to within $2-\epsilon$," *J. Comput. Syst. Sci.*, vol. 74, pp. 335–349, 2008.

[62] Kleywegt, A. J. and Papastavrou, J. D., "The dynamic and stochastic knapsack problem," *Operations Research*, vol. 46, no. 1, pp. 17–35, 1998.

[63] Könemann, J., Pritchard, D., and Tan, K., "A partition-based relaxation for Steiner trees," *CoRR*, vol. abs/0712.3568, 2007. informal publication.

[64] Kriesell, M., "Local spanning trees in graphs and hypergraph decomposition with respect to edge connectivity," *Technical Report 257, University of Hannover*, 1999.

[65] Kriesell, M., "Edge-disjoint trees containing some given vertices in a graph," *J. Combin. Theory. Series B*, 2003.

[66] Lau, L. C., "Packing Steiner forests," *Proceedings of IPCO*, pp. 362–376, 2005.

[67] Lau, L. C., "An approximate max-Steiner-tree-packing min-Steiner-cut theorem," *Combinatorica*, vol. 27, no. 1, pp. 71–90, 2007.

[68] Lau, L. C., Naor, S., Salavatipour, M., and Singh, M., "Survivable network design with degree or order constraints.," *Proceedings of* 39^{th} *Annual ACM Symposium on the Theory of Computing (STOC)*, pp. 651–660, 2007.

[69] Lau, L. C. and Singh, M., "Additive approximation for bounded degree survivable network design," *To appear in Proceedings of* 40^{th} *Annual ACM Symposium on the Theory of Computing (STOC)*, 2008.

[70] Lehmann, B., Lehmann, D., and Nisan, N., "Combinatorial auctions with decreasing marginal utilities," *Proceedings of* 3^{rd} *ACM Conference on Electronic Commerce (EC)*, pp. 18–28, 2001.

[71] Lenstra, J. K., Shmoys, D., and Tardos, E., "Approximation algorithms for scheduling unrelated parallel machines," *Math. Programming*, vol. 46, pp. 259–271, 1990.

[72] Li, Z. and Li, B., "Efficient computation of maximum multicast rate," *Proceedings of* 22^{nd} *Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, pp. 1618–1628, 2003.

[73] Lueker, G. S., "Average-case analysis of off-line and on-line knapsack problems," *Journal of Algorithms*, vol. 29, no. 2, pp. 277–305, 1998.

[74] Mader, W., "A reduction method for edge-connectivity in graphs," *Ann. Discrete Math.*, vol. 3, pp. 145–164, 1978.

[75] Marchetti-Spaccamela, A. and Vercellis, C., "Stochastic on-line knapsack problems," *Mathematical Programming*, vol. 68, pp. 73–104, 1995.

[76] Megiddo, N., "Combinatorial optimization with rational objective functions," *Mathematics of Operations Research*, vol. 4(4), pp. 414–424, 1979.

[77] Mehta, A., Saberi, A., Vazirani, U. V., and Vazirani, V. V., "Adwords and generalized on-line matching," *Journal of the ACM*, vol. 54, no. 5, pp. 1–22, 2007.

[78] Nash-Williams, C. S. J. A., "Edge disjoint spanning trees of finite graphs," *J. Lond. Math. Soc.*, 1961.

[79] Nguyen, C., Shen, J., Hou, M. M., Sheng, L., Miller, W., and Zhang, L., "Approximating the spanning star forest problem and its applications to genomic sequence alignment," *Proceedings of* 17^{th} *Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pp. 645–654, 2007.

[80] Noga, J. and Sarbua, V., "An online partially fractional knapsack problem," in *Proc. 8th Sym. Parallel Architectures, Algorithms and Networks*, pp. 108–112, 2005.

[81] Papastavrou, J. D., Rajagopalan, S., and Kleywegt, A. J., "The dynamic and stochastic knapsack problem with deadlines," *Management Science*, vol. 42, no. 12, pp. 1706–1718, 1996.

[82] Provan, J. S. and Ball, M. O., "The complexity of counting cuts and of computing the probability that a graph is connected.," *SIAM J. Computing*, vol. 12, pp. 777–788, 1983.

[83] Raghavan, P. and Thomson, C. D., "Randomized rounding: a technique for provably good algorithms and algorithmic proofs.," *Combinatorica*, vol. 7, pp. 365–374, 1987.

[84] Rajagopalan, S. and Vazirani, V., "On the bidirected cut relaxation for the metric Steiner tree problem," in *Proceedings of the tenth annual ACM-SIAM symposium on Discrete algorithms*, pp. 742–751, 1999.

[85] Rizzi, R., "On Rajagopalan and Vazirani's 3/2-approximation bound for the iterated 1-Steiner heuristic.," *Information Processing Letters*, vol. 86, pp. 335–338, 2003.

[86] Robins, G. and Zelikovsky, A., "Tighter bounds for graph Steiner tree approximation.," *SIAM Journal of Discrete Mathematics*, vol. 19, pp. 122–134, 2005.

[87] Sandholm, T. and Suri, S., "Side constraints and non-price attributes in markets," *Games and Economic Behavior*, vol. 55, no. 2, pp. 321–330, 2006.

[88] Schrijver, A., *Combinatorial Optimization.* Springer-Verlag, Berlin, 2003.

[89] Shmoys, D. and Tardos, É., "An approximation algorithm for the generalized assignment problem," *Math. Programming*, vol. 62, pp. 461–474, 1993.

[90] Singh, M. and Lau, L. C., "Approximating minimum bounded degree spanning trees to within one of optimal.," *Proceedings of* 39^{th} *Annual ACM Symposium on the Theory of Computing (STOC)*, pp. 661–670, 2007.

[91] Tutte, W. T., "On the problem of decomposing a graph into n connected factors," *J. Lond. Math. Soc.*, 1961.

[92] VAN SLYKE, R. and YOUNG, Y., "Finite horizon stochastic knapsacks with applications to yield management," *Oper. Res.*, vol. 48, pp. 155–172, 2000.

[93] VAZIRANI, V. V., *Approximation Algorithms*. Springer, 2002.

[94] VONDRÁK, J., "Optimal approximation for the submodular welfare problem in the value oracle model," *To appear in Proceedings of* 40^{th} *Annual ACM Symposium on the Theory of Computing (STOC)*, 2008.

[95] WONG, R. T., "A dual ascent approach for Steiner trees on a directed graph," *Mathematical Programming*, vol. 28, pp. 271–287, 1984.

[96] YAO, A. C.-C., "Probabilistic computations: towards a unified measure of complexity," in *Proc. 18th IEEE Foundations of Computer Science (FOCS)*, pp. 222–227, 1977.