Aspects of Network Design

Adriana Karagiozova

A Dissertation
Presented to the Faculty
of Princeton University
in Candidacy for the Degree
of Doctor of Philosophy

Recommended for Acceptance
by the Department of
Computer Science

November, 2007

Abstract

In this dissertation we study two problems from the area of network design.

The first part discusses the multicommodity buy-at-bulk network design problem, a problem that occurs naturally in the design of telecommunication and transportation networks. We are given an underlying graph and associated with each edge of the graph, a cost function that represents the price of routing demand along this edge. We are also given a set of demands between pairs of vertices each of which needs to be satisfied by paying for sufficient capacity along a path connecting the vertices of the pair. In the multicommodity network design problem the objective is to minimize the cost of satisfying all demands.

There are often situations where there is an initial fixed cost of utilizing an edge, or there is discounting or economies of scale that give rise to concave cost functions. We have an instance of the buy-at-bulk network design problem when the cost functions along all edges are concave.

Unlike the case of linear cost functions, for which polynomial time algorithms exist, the buy-at-bulk network design problem is NP-hard. We give the first non-trivial approximation algorithm for the general case of the problem with arbitrary concave costs along the edges. Our algorithm is conceptually very simple and has an approximation guarantee of $e^{O(\sqrt{\log n \log\log n})} \log D$, where n is the number of demand pairs and D is the maximum demand.

The second part of the thesis examines the *Terminal Backup* problem that arises when facilities storing data would like to be connected to at least one other facility for data backup purposes. In the *Terminal Backup* problem we are given a graph with terminal nodes, non-

terminal nodes and edge costs. The objective is to find the cheapest forest connecting every terminal to at least one other terminal. We show that this problem is reducible to *Simplex Matching*, a variant of 3D matching that we introduce.

In an instance of *Simplex Matching*, we are given a hypergraph with edges of size at most three and edge costs associated with them. We show how to find the minimum cost perfect matching of this hypergraph efficiently if the edge costs obey a simple and realistic inequality we call the *Simplex Condition*.

While motivated by the desire of solving the *Terminal Backup* problem, the usefulness of the *Simplex Matching* algorithm we develop is not limited to *Terminal Backup*. We briefly discuss additional problems where *Simplex Matching* can be successfully employed.

Acknowledgements

First and foremost, I would like to thank my advisor Moses Charikar. Moses has been an inspiration, a great mentor and a friend throughout my graduate career. I am very grateful for his guidance, encouragement and for letting me work on whatever problems seemed interesting even if they were not on the same topic.

I would also like to thank the members of my thesis committee: Bernard Chazelle and Bob Tarjan, who read my thesis and gave helpful suggestions, and Bob Sedgewick and Boaz Barak for useful comments at the presentation of my thesis proposal.

I would like to thank my collaborators for their contribution to this work. Part I of this dissertation is joint work with Moses Charikar that appeared in STOC 2005. Part II is joint work with Elliot Anshelevich that appeared in STOC 2007. I am very grateful for Elliot's perseverance in attacking the *Simplex Matching* problem for months even when it did not yield. Special thanks to Paul Seymour for providing most of the proof of the non-planar case of Lemma 5.4.7. Thanks to Uri Zwick for his questions on the unweighted version of *Simplex Matching*, which prompted me to include it in the thesis.

I would also like to thank my other collaborators, Bo Brinkman and James Lee, with whom I worked on embedding series-parallel graphs in ℓ_1, work that is not included in this thesis. Bo, it was a real pleasure working with you and I hope we get the chance to work together in the future.

Princeton has been a very exciting place for someone interested in Computer Science Theory. I greatly enjoyed and benefitted from numerous talks at theory lunch and at the seminars at IAS. The computer science department has been a very welcoming place. In

particular, many thanks to Melissa Lawson and Mitra Kelly for handling all administrative tasks and letting us concentrate on research.

I would like to thank my friends at the department for making the breaks from work fun, for lots of nice conversations and afternoon tea. I especially want to thank Kevin Wayne for always welcoming intrusions in his office, taking care of my plants and being a good friend.

My graduate years would never have been the same without the focus and intensity of the Princeton Shotokan Karate practices. Special thanks to Margaret Lo, my sensei and friend, for pushing me to do better and for believing in me.

To my best friend Julie Viewheg: thank you for being there when I needed you.

Last but not least, I would like to thank my family for their love and support throughout the years. My parents understand that doing research is not easy and have always stood by me and encouraged me. Most of all, I would like to thank my husband Davide Gaiotto for his love and unwavering support all through my years in graduate school. His kindness and patience are unparalleled and he has always believed in my capacity as a researcher, a wife and a mother. Finally, I would like to thank my son Corin for all the sweet hugs he gave me while I was writing this thesis and for being my infinite source of happiness.

Adriana Karagiozova
Princeton University
October 2007

I gratefully acknowledge funding by a Gordon Wu fellowship and through Moses Charikar's NSF grants CCR-0205594, CCR-0237113, DOE award DE-FG02-02ER25540, MSPA-MCS award 0528414 and Alfred P. Sloan Fellowship.

Contents

List of Figures

List of Tables

Part I

Multicommodity Buy-at-Bulk Network Design

Chapter 1

Introduction

Consider a postal company that owns several types of trucks and airplanes and is in the business of transporting packages between various locations. The characteristics of each package are naturally a source and a destination as well as size that may be weight or dimensions. The task that the company is faced with on a daily basis is deciding how to route the packages so that the total routing cost is minimized.

A small truck might pick up a package at the sender's home and drive it to the nearest local shipping facility, where it would be loaded into a larger truck together with other packages of similar destination and driven to the airport. At the airport the contents of this truck and possibly others will be moved to a plane and sent to the next stop of the route to the destination. Some of the packages may then be unloaded into a large truck and driven to the local shipping facility near the destination of the package we started with and from there into a small truck that would make a delivery at the final destination.

At each segment of the route of a package the company has a choice of using a combination of different types of trucks and possibly airplanes to transport all packages along this segment. Typically it is cheaper to use a larger truck than several small trucks of the same combined capacity and thus there are economies of scale to be captured by aggregating traffic bound for similar destinations.

Now consider a scenario of similar flavor that occurs when designing telecommunication

networks to route a given set of demands. There are several cable types available with each type having a fixed per length cost and some capacity. We need to lay a collection of cables along some of the edges of the network in order to install sufficient capacity to route all demands. Bandwidth experiences significant economies of scale so the cost of routing a unit of demand along a cable with higher capacity is less than if a lower capacity cable was used and thus it is cost effective to aggregate traffic over large backbone links [1]. The objective is to design a minimum cost network that allows all demands to be routed.

Notice that in both examples the function corresponding to the minimum cost of routing demand along a given segment or edge is a sub-additive monotone function, reflecting the existing economies of scale. Both scenarios as instances of the multicommodity buy-at-bulk network design problem in which we seek to design a minimum cost network that satisfies the demands between terminals from a given set of source-sink pairs such that the cost functions of routing demand along the edges are sub-additive monotone functions.

1.1 Formal Definition of the Problem

The multicommodity buy-at-bulk network design problem is defined as follows. We are given an underlying graph $G(V, E)$ and a set of source-sink pairs $\mathcal{D} = \{(s_i, t_i) |\ s_i, t_i \in V\}$, $|\mathcal{D}| = n$. Associated with each edge e in the graph are a positive length l_e and a sub-additive monotone function $f_e : \mathbb{Z}^+ \to \mathbb{R}^+$ that gives the cost (per unit length of e) of transporting demand along e. Each source-sink pair has a demand given by the function $\delta : D \to \mathbb{Z}^+$. The goal is to design a minimum cost network that carries $\delta(s_i, t_i)$ units of demand from s_i to t_i for all demand pairs (s_i, t_i). The cost of a feasible solution is determined by the amount of demand routed along each edge. Let x_e denote the amount routed along edge e. Then the total cost of the solution is

$$\sum_{e \in E} f_e(x_e)\ l_e$$

The problem can be classified according to two criteria. We say that the problem is *uniform* when the cost functions along the edges are identical, i.e. $f_e = f$ for all $e \in E$; otherwise we say it is *non-uniform*. Notice that in the non-uniform version the length of an edge can be incorporated in its cost function so we can assume that $l_e = 1$ for all $e \in E$ without loss of generality. We refer to the problem as *single-source* when all source-sink pairs share the same source terminal. When the source and sink terminals can be arbitrary vertices in the graph, we refer to the problem as *multi-source*.

1.2 Brief Literature Survey

The multicommodity buy-at-bulk network design problem arises commonly in practical network design, where the aim is to determine where one should buy capacity in order to satisfy a set of demands. Situations in which there is a start-up cost, discounting or economies of scale give rise to concave cost functions along the edges of the network [76]. Real world problems from a variety of industries can be modeled as multicommodity buy-at-bulk network design problems. The areas of application include telecommunications, transportation, airline route planning, production planning, water and oil resource management, etc. [62] The practical importance of the problem has made it a target of extensive study in the Operations Research literature since the *70*'s [13, 35, 38, 39, 76]. Salman et al. [67] give an overview of the early work on the problem, which for the most part concentrates on heuristics or on characterizing the optimal solutions of the problem and subsequently solving small instances exactly.

1.2.1 Early Algorithmic Studies

One of the early algorithmic studies of the multicommodity buy-at-bulk problem was initiated by Salman et al. [67] They consider the single-source version of the problem where the input consists of an undirected graph with one source and multiple sinks, each sink v with a specified demand dem_v. Capacity on a link can be purchased in discrete units (cables) $u_1 < u_2 < \ldots < u_r$ with costs $c_1 < c_2 < \ldots < c_r$ such that the cost per unit of

capacity decreases $c_1/u_1 > c_2/u_2 > \ldots > c_r/u_r$. They show that this problem is NP-hard via reductions from both Steiner Tree and Knapsack. The reductions show two inherent sources of hardness of the problem – connectivity (the problem is NP-hard even when only one cable type is available) and choice of cables (the problem is NP-hard even when the graph consists of a single edge). As it is unlikely that there exist polynomial time algorithms that would solve the problem exactly, they turn to approximation algorithms.

Definition 1.2.1 *An α-approximation algorithm for a minimization problem is an algorithm that for any instance of the problem returns in polynomial time a feasible solution with objective value at most a factor of α greater than the optimum for this instance.*

Salman et al. [67] give an $O(\log(D/u_1))$-approximation algorithm for the Euclidean case of the single-source multicommodity buy-at-bulk problem considered, where the vertices are represented as points in the plane and the length function on the edges is the Euclidean distance (here $D = \sum_v dem_v$ is the total demand). Additionally, they give an $O(\log n)$-approximation algorithm when the metric is arbitrary and every source-sink path is restricted to at most two edges, where n is the number of vertices in the underlying graph.

Mansour and Peleg [57] consider another special case of the problem in which there are multiple sources and sinks but only one type of cable with a basic unit of capacity. Installing any edge has a fixed cost that is the same across all edges, as well as a variable cost per unit of capacity installed that is edge-dependent. They give an $O(\log n)$-approximation algorithm for this variant (or constant in the Euclidean case [67]).

1.2.2 Multi-source Uniform Case

Awerbuch and Azar [12] study the same framework as Salman et al. [67] and devise approximation algorithms that decouple the tasks of selecting the routes that connect each source-sink pair and selecting the cables to be installed on each edge. If $f_e(x_e)$ is the minimum cost of routing x_e units along the edge e, they note that even if computing f_e exactly is an integer min-knapsack problem that is NP-hard [56], a 2-approximation can be found

in linear time. Consequently it is sufficient to determine how demand should be routed first and then decide what cables to install along each edge.

To determine how to route the demands they use probabilistic metric approximation, namely the fact that any metric can be α-probabilistically approximated by a set of tree metrics over the same set of points. This means that there exists a polynomially computable probability distribution over trees with the following two properties: distances between pairs of points do not shrink in any of the trees in the distribution and expected distances over the distribution on the tree metrics do not stretch by more than a factor of α. Using only the fact that the edge cost functions are identical (and sub-additive) they define the following randomized α-approximation algorithm: choose a random tree from the distribution that α-probabilistically approximates the underlying graph metric of the problem and route the demands along the unique paths between source-sink pairs in the selected tree. At the time of publication, the best-known value of α was $O(\log^2 n)$ due to Bartal [14], later improved to $O(\log n \log\log n)$ by Bartal [15] and finally to $O(\log n)$ by Fakcharoenphol et al. [34], where the construction is also derandomized. This results in the current best approximation algorithm for the multi-source uniform version of the buy-at-bulk problem with approximation guarantee of $O(\log n)$.

1.2.3 Single-source Uniform Case

The single-source uniform version of the problem has been extensively studied. Andrews and Zhang [6] define the *access network design* problem, a special case of the single-source uniform buy-at-bulk problem where the cost of installing capacity on the edges is a concave piecewise linear function with K breakpoints. This corresponds to the scenario when K different trunk-types are available, each having a start-up cost and a per unit cost of carrying traffic between nodes in the network. They give an $O(K^2)$-approximation algorithm for this problem. The approximation factor was later improved to $O(K)$ by Garg et. al [36] and independently brought down to a constant by Guha et al. [40]

The general single-source uniform version of the problem can be approximated up to a

constant [1] . Guha et al. [41] and Gupta et al. [43] give a randomized combinatorial constant factor approximation algorithm that constructs a layered solution such that at every layer the demand of a chosen set of source-sink pairs is satisfied at a fixed per unit cost. Talwar [72] utilizes an LP rounding approach to give another constant factor approximation algorithm and show that a natural LP formulation of the problem has a constant integrality gap[2] .

1.2.4 Single-source Non-uniform Case

At the time of publication of the result that forms the basis of the first part of this dissertation [21], the only non-uniform version of the problem with a non-trivial approximation algorithm was the single-source version. Meyerson et al. [59] present the *Cost-Distance* problem where the goal is to find a Steiner tree that optimizes the sum of edge costs along one metric and the sum of source-sink distances along an unrelated metric. They reduce the non-uniform single-source buy-at-bulk problem to *Cost-Distance* by noticing that if each edge is replaced with a suitable collection of edges each of which has a fixed cost and an incremental cost of routing demand, then the optimum solution with the new edges is no more than a factor of 2 away from the optimum to the original problem (this observation is implicit in [12, 67]). They give a randomized combinatorial algorithm for *Cost-Distance* with approximation guarantee of $O(\log k)$, where k is the number of source-sink pairs in the instance. The algorithm proceeds by pairing up sinks (or the source and a sink) until only the source remains. At each stage it finds a matching between the source and sinks, chooses one vertex from each matched pair and transports all the demand from the other vertex in the pair to the chosen one at a cost that is no more than a constant factor away from the cost of the optimal solution. This results in a randomized $O(\log k)$-approximation algorithm for the non-uniform single-source case, later derandomized by Chekuri et al. [23] by using an LP relaxation.

The analysis of the *Cost-Distance* algorithm from [59] uses the tree structure of the

[1] In other words there exists a constant factor approximation algorithm for it.

[2] If A' is an LP relaxation of a minimization problem A and $A(I)$ and $A'(I)$ denote the optimal value of A and A' respectively on an instance I, then the integrality gap of the LP formulation A' is the minimum over all instances I of $A(I)/A'(I)$.

optimal solution in a crucial way and seems difficult to modify to accommodate the more complex structure of the optimal solution to the multi-source version of the problem.

1.2.5 Rent-or-Buy Problem

An interesting special case of the multi-source uniform problem is the *Rent-or-Buy* problem, where one can either rent capacity along each edge by paying per unit of capacity or buy unlimited bandwidth at a large fixed cost M, resulting in a cost function of the form $f(x) = \min\{x, M\}$. The problem was first studied by Awerbuch et al. [11] in the online setting (referred to as the *network connectivity leasing problem*) where they gave an $O(\log^2 n)$-competitive algorithm by modifying a simple combinatorial algorithm for the Generalized Steiner Tree problem. Bartal et al. [16] consider the network connectivity leasing problem as the relaxed task systems corresponding to the generalized Steiner tree problem and give an $O(\log n)$-competitive algorithm.

The first constant factor algorithm for the rent-or-buy problem is due to Kumar et al. [53] via a primal-dual algorithm. As they remark, the rent-or-buy problem seems to capture the "route vs. gather" essence of the multicommodity buy-at-bulk network design problem. Without knowledge of where demand could be aggregated any given source-sink pair would try to route its demand along the edges of the shortest path between the two vertices, renting capacity along them. The presence of other source-sink pairs makes aggregating flow on some edges very profitable by buying a large quantity of capacity at a low per unit cost.

Gupta et al. [42] give a simple improved constant factor approximation algorithm for the multicommodity rent-or-buy problem. They mark each source-sink pair with probability $\frac{1}{M}$ and buy the edges of an approximate Steiner forest connecting the marked pairs. The rest of the pairs are connected via the shortest path in the graph given the already bought edges. This conceptually simple algorithm provided some of the inspiration for our Inflated Greedy algorithm. An essential part of Gupta et al.'s solution is a primal-dual type algorithm for the generalized Steiner problem that allocates the total cost of building the

Steiner forest to the source-sink pairs in a "fair" manner (a strict cost-sharing method), making sure that no pair imposes a large burden on building the network but pays only a small fraction of the total cost.

1.2.6 Our Contribution

At the time of publication of the result that forms the basis of the first part of this dissertation [21], the non-uniform multicommodity buy-at-bulk network design problem was the one remaining problem in buy-at-bulk network design for which no non-trivial approximation results were known. The existing techniques for the single-source case (repeatedly merging demand) and the uniform case (using approximations by tree metrics) fail in the general case. Since there are multiple sources, the structure of the optimal solution can be quite complex, making it difficult to merge routes by demand pairs. At the same time, since every edge has a different cost function, there is no useful notion of length of edges to enable the application of tree metrics.

We presented the first approximation algorithm to the multicommodity buy-at-bulk network design problem in the general case. Our algorithm is an extremely simple randomized greedy algorithm. In the non-uniform multi-source case it yields an approximation factor of $e^{O(\sqrt{\log n \log\log n})} \log D$, where n is the number of demand pairs and D is the maximum demand. When our algorithm is applied to the non-uniform single-source version of the problem, we get an approximation factor of $O(\log^2 n) \log D$. It is interesting that when applied to the single-source case, the algorithm achieves approximation ratio that is close to that of the best known algorithm ($O(\log n)$ from [23, 59]). Our approach is reminiscent of the greedy online algorithm for generalized Steiner tree by Awerbuch, Azar and Bartal [11] as well as the techniques used by Meyerson [58] for designing online algorithms for network design problems.

1.2.7 Subsequent Results

As mentioned in Charikar and Karagiozova [21], one of the most interesting problems to tackle was whether the approximation factor can be reduced to *polylog*(n), a possibility that was not ruled out by the known hardness results at the time [5]. Another open problem was to remove the dependence of the approximation factor on the *maximum demand* and obtain a bound only in terms of the number of source-sink pairs. Both of these open problems were resolved by Chekuri et al. [22] who give an approximation algorithm for the general case of the problem of $O\left(\min\{\log^3 n \log D, \log^5 n \log\log n\}\right)$. The algorithm proceeds iteratively, constructing a partial solution of low density that connects some of the remaining source-sink pairs. The main idea behind constructing such a low density partial solution lies in showing the existence of one with a specific structure (junction-tree). They present two algorithms for computing such a junction-tree: an LP relaxation for the case of arbitrary demands and a greedy combinatorial algorithm based on computing shallow-light trees when the maximum demand D is polynomial in n. For more details see Chapter 3 and [22].

1.2.8 Hardness Results

A natural question that occurs in the study of the buy-at-bulk network design problem is how low we can hope to push the approximation factors for the various versions of the problem. Andrews [5] shows that for any constant $\gamma > 0$ there is no $O(\log^{1/2-\gamma} n)$-approximation algorithm for the non-uniform version and no $O(\log^{1/4-\gamma} n)$-approximation algorithm for the uniform version unless $NP \subseteq ZPTIME(n^{\text{polylog}(n)})$ [3] . Chuzhoy et al. [25] show that no $O(\log\log n)$-approximation algorithm exists for the single-source version under the same complexity assumption.

[3]Recall that $ZPTIME(n^{\text{polylong}(n)}) = RTIME(n^{\text{polylong}(n)}) \cap coRTIME(n^{\text{polylong}(n)})$ and equals the class of problems solvable by a randomized algorithm that always returns the correct answer and has expected running time $O(n^{\text{polylog}(n)})$. [5]

Problem variant	**Result**	**Paper**	**Year**
single-source uniform, Euclidean	$O(\log D)$	[67]	1997
single-source uniform, path length ≤ 2	$O(\log n)$	[67]	1997
multi-source uniform, one type of cable	$O(\log n)$	[57]	1994
multi-source uniform, one type of cable, Euclidean	$O(1)$	[67]	1997
multi-source uniform	$O(\log^2 n)$	[12, 14]	1997
multi-source uniform	$O(\log n \log\log n)$	[12, 15]	1998
multi-source uniform	$O(\log n)$	[12, 34]	2003
access network design	$O(K^2)$	[6]	1998
access network design	$O(K)$	[36]	2001
access network design	$O(1)$	[40]	2000
single-source uniform	$O(1)$	[41]	2001
single-source uniform	$O(1)$	[43]	2003
single-source uniform	$O(1)$	[72]	2002
single-source non-uniform	$O(\log n)$	[59, 23]	2000
network connectivity leasing	$O(\log^2 n)$	[11]	1996
network connectivity leasing	$O(\log n)$	[16]	1997
rent-or-buy	$O(1)$	[53]	2002
rent-or-buy	$O(1)$	[42]	2003
multi-source non-uniform	$e^{O(\sqrt{\log n \log\log n})} \log D$	[21]	2005
multi-source non-uniform	$O(\log^3 n \log D)$	[22]	2006
multi-source non-uniform	$O(\log^5 n \log\log n)$	[22]	2006
multi-source uniform, hardness	$\Omega(\log^{1/4-\gamma} n)$	[5]	2004
multi-source non-uniform, hardness	$\Omega(\log^{1/2-\gamma} n)$	[5]	2004
single-source non-uniform, hardness	$\Omega(\log\log n)$	[25]	2005

Table 1.1: Buy-at-bulk approximation results

Chapter 2

The Inflated Greedy Algorithm

In this chapter we present Inflated Greedy – a randomized greedy algorithm for instances of the multi-source non-uniform buy-at-bulk network design problem where each source-sink pair needs to route a single unit of demand. The algorithm is conceptually very simple and easy to implement as it only performs repeated shortest paths calculations. We show that it has an $e^{O(\sqrt{\log n \log\log n})}$ approximation guarantee, where n is the number of source-sink pairs in the graph.

2.1 Inflated Greedy Description

The algorithm starts off by permuting the source-sink pairs and assigning an inflated demand to each pair as a function of its position in the permutation. It proceeds by constructing a network in a greedy manner, going down the list of the source-sink pairs and connecting each in the cheapest way possible.

Inflated Greedy **Algorithm:**

1. Randomly permute the source-sink pairs.

   Let $\pi = ((s_1, t_1), (s_2, t_2), \ldots, (s_n, t_n))$ be the chosen permutation.

2. Define a new demand function $\delta' : \mathcal{D} \to \mathbb{R}^+$ such that $\delta'(s_k, t_k) = \frac{n}{k}$.

3. For all $k = 1, \ldots, n$ in that order, connect s_k to t_k via the path with the smallest cost of routing $\delta'(s_k, t_k)$ units of demand, using the network constructed for the previous $k - 1$ pairs.

4. Route a single unit of demand between s_k and t_k along the path selected at the k-th iteration of Step 3.

This algorithm is somewhat reminiscent of the algorithm of Gupta et al. [42] for the rent-or-buy problem. Recall that the rent-or-buy problem is a special case of the multi-source uniform buy-at-bulk network design problem where the cost functions along the edges are of the form $f(x) = \min\{x, M\}$, where M is usually very large. This means that unlimited capacity can be bought on each edge for a price of M or capacity can be rented at a fixed price per unit of demand routed along the edge. The algorithm of Gupta et al. takes as input an instance of the rent-or-buy problem with unit demands for each source-sink pair. It starts by marking each pair with probability $\frac{1}{M}$ and then buys the edges that connect all the marked pairs. The source and sinks of any unmarked pairs are then connected greedily via the shortest path in the graph. Notice that if a pair is marked we can think of its demand being inflated by a factor of M in which case it pays off to buy the edges that connect the source and sink of the pair. Any other pair that decides to use already bought edges would not have to pay anything for routing capacity along them.

A similar situation occurs with the Inflated Greedy algorithm. By randomly permuting the source-sink pairs at the beginning we are performing a step like the marking process in Gupta et al.'s rent-or-buy algorithm. The first pair in the permutation has its demand inflated to n so when we buy sufficient capacity along some edges to satisfy this demand, any subsequent pairs using those connecting edges would need to pay only a small fraction of what the first pair paid to satisfy their inflated demands. After the k-th pair is connected we have a forest along each edge of which we have paid for at least $\frac{n}{k}$ units of capacity and any subsequent pairs using those edges will need to pay only a small fraction of what has already paid by the previous pairs.

The key challenge to solving the multicommodity buy-at-bulk problem is to decide which

edges should be bought. Intuitively, we would like to buy edges that many source-sink pairs would use to route their demand in a cheap way. However it seems difficult to decide which edges would be useful. The approach taken by Inflated Greedy is based on the motivation that if we had a representative random sample of source-sink pairs then the edges required to connect the pairs in the sample are likely to prove useful for connecting other source-sink pairs. Inflated Greedy processes the source-sink pairs in random order and we can think of a size k prefix of the permutation π as a random sample of k source-sink pairs. If this sample is to reflect the properties of the n source-sink pairs we have to make sure that the total demand of the sample matches the total demand n and thus inflate the demands of all pairs in the sample by a factor of $\frac{n}{k}$. In Step 2 of Inflated Greedy we attempt to simulate this by assigning demand $\frac{n}{k}$ to the k-th pair in the permutation.

When proving the performance guarantee of Inflated Greedy we will estimate the cost of satisfying the inflated demands with respect to the optimal cost of satisfying the original unit demands. At first glance it is not immediately clear why inflating the demands by as much as Inflated Greedy does would not dramatically increase the cost. Notice, however, that for any source-sink pair the demand gets inflated to $\frac{n}{i}$ with probability $\frac{1}{n}$ for all $i = 1, 2, \ldots, n$, so on average the demand of each pair gets inflated by a factor of $\ln n$. Thus the expected cost of the optimal solution of satisfying the inflated demands would be a factor of $\ln n$ away from the cost of the optimal solution of satisfying the original unit demands since the cost functions along the edges are sub-additive.

2.2 Running Time

Connecting the k-th pair (s_k, t_k) in Step 3 of Inflated Greedy involves a simple shortest-path calculation. To each edge e of the graph, we assign a length that is equal to the cost of transporting additional $\frac{n}{k}$ units of demand along it. Then we compute the shortest path between s_k and t_k according to this length function. Notice that if the edge e has not been used before, the length assigned to it is simply equal to $f_e\left(\frac{n}{k}\right)$. Otherwise if x_e units of demand have already been routed along it, the length is $f_e\left(x_e + \frac{n}{k}\right) - f_e\left(x_e\right)$.

This means that the algorithm only performs n shortest path calculations between two vertices in the graph, resulting in a running time of $O\left(n(|E| + |V| \log |V|)\right)$.

2.3 Analysis of Inflated Greedy

We will estimate the cost of satisfying the inflated demands, which is an upper bound on the price we need to pay to route the unit demands once the paths have been selected (the edge cost functions are monotone). Our goal is to prove that when Inflated Greedy selects a path to connect the pair (s_k, t_k), it does so in a relatively cheap way. Instead of bounding directly the cost of the shortest path selected in Step 3 of the algorithm, we will show that there exists a path connecting (s_k, t_k) of specific structure, called an *alternating path*, that costs relatively little to route $\frac{n}{k}$ units of demand on. Since Inflated Greedy selects the cheapest path to route $\frac{n}{k}$ units of demand, a bound on the cost of such alternating path automatically gives a bound on the cost of the path Inflated Greedy selects in Step 3.

2.3.1 Preliminaries

The path connecting s_k to t_k found by Inflated Greedy consists of segments that have never been used before as well as segments that are part of the already constructed network. Moreover, Inflated Greedy selects the cheapest path with this structure. We consider a collection of paths (*alternating paths*) that is a subset of what Inflated Greedy can choose from when connecting s_k to t_k in Step 3 of the algorithm.

Definition 2.3.1 *An* alternating path *connecting s_k to t_k is a path consisting of alternating segments $S_1, S'_1, S_2, S'_2, \ldots, S_g, S'_g, S_{g+1}$. The segments S_i are paths along edges used in the optimal solution to the original unit demands problem and the segments S'_i are the complete paths constructed in Step 3 of* Inflated Greedy *for demand pairs that preceded (s_k, t_k) in π (see Figure 2.1).*

Definition 2.3.2 *The* hops *of an alternating path $S_1, S'_1, S_2, S'_2, \ldots, S_g, S'_g, S_{g+1}$ are the segments $S'_1, S'_2, \ldots, S'_g$.*

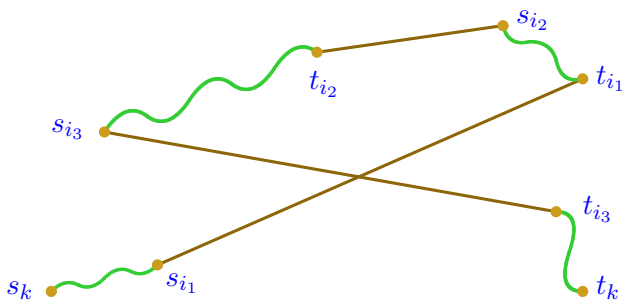


Figure 2.1: An alternating path connecting s_k to t_k. The straight line segments correspond to the *hops*, the wavy segments consist of edges from the optimal solution ($i_1, i_2, i_3 < k$).

The key idea is to show that there are cheap alternating paths connecting s_k to t_k. In order to obtain an alternating path of low cost it is important to use only a few of the available $k-1$ paths as we would be paying for routing additional $\frac{n}{k}$ units of demand along each of them. We will restrict our attention to alternating paths connecting s_k to t_k with at most g_k hops (see Definition 2.3.2); g_k is a parameter whose value will be specified later. We will construct an auxiliary graph with vertices corresponding to contracted paths used by the optimal solution and edges corresponding to a selection of the pairs (s_i, t_i) for $i \leq k$. Then a cycle in this graph containing the edge that corresponds to (s_k, t_k) can be viewed as a contracted version of an alternating path that connects s_k and t_k. There is a tradeoff between the cost of the optimal solution path that was contracted at each vertex of the auxiliary graph and its girth[1] : the smaller the cost of the path corresponding to each vertex, the larger the girth of the graph and vice versa. We will show that when we place an upper bound on the cost at each vertex, the girth of the auxiliary graph is still sufficiently small. This argument is reminiscent of the argument in [11] for online generalized Steiner tree. In the Steiner tree case, once edges are bought, additional pairs can use them for free. However, our argument is much more complicated by the fact that we may have to pay an additional amount for edges that have already been used before.

[1]Recall that the girth of a graph is the length of its shortest cycle.

2.3.2 Additional Notation

Let $R = \{(s_1, t_1), (s_2, t_2), \ldots, (s_k, t_k)\}$ be the set of the first k source-sink pairs in the permutation π. Let C_k^R be the cost (as a function of R) of connecting the k-th pair in Step 3 of the algorithm by routing $\delta'(s_k, t_k)$ units of demand along an alternating path with at most g_k hops. Let $C_k = \mathrm{E}_\pi\left[C_k^R\right]$, where the expectation is taken over the random permutation π. If T_n is the total cost incurred by Inflated Greedy of connecting all source-sink pairs in $\mathcal{D}$ while satisfying the inflated demands then $\mathrm{E}_\pi[T_n] \leq \sum_{k=1}^{n} C_k$. Naturally, T_n is an upper bound on the cost of satisfying the original unit demands along the paths selected in Step 3, as $\delta'(s_k, t_k) \geq 1$ and the edge cost functions are sub-additive. Finding an upper bound on $\mathrm{E}_\pi[T_n]$ directly results in a bound on the expected cost of routing the original unit demands.

Let α_k^R denote the cost of routing $\frac{n}{k}$ units of demand along the portion of the alternating path that corresponds to edges in the optimal solution, given that R is the set of the first k source-sink pairs. Let β_k^R be the cost of routing $\frac{n}{k}$ units of demand along the hops of the alternating path, again given that R is the set of the first k source-sink pairs. Then $C_k^R = \alpha_k^R + \beta_k^R$. In the following subsections we will describe how to construct a good path and bound the expected value of α_k^R and β_k^R corresponding to it. Note that throughout the analysis we rely heavily on the randomization introduced by picking a random permutation of the source-sink pairs and most of the quantities we bound are expectations of costs.

2.3.3 Bounding $\mathrm{E}_\pi\left[\alpha_k^R\right]$

Consider how the first k source-sink pairs are connected in the optimal solution: let the connected components of the union of the paths used to connect the elements of R be $A_1, A_2, \ldots, A_l$ where $1 \leq l \leq k^2$ [2] (see Figure 2.2). For all $i = 1, \ldots, l$ let OPT_i^R be the cost of routing $\frac{n}{k}$ units of demand along all edges of A_i and let k_i be the number of source-sink pairs served in A_i, where $k_i > 0$ and $\sum_{i=1}^{l} k_i = k$.

We will now argue that $\mathrm{E}_\pi\left[\sum_i OPT_i^R\right]$ is bounded from above by the cost of the optimal

[2] Strictly speaking, l is a function of R.

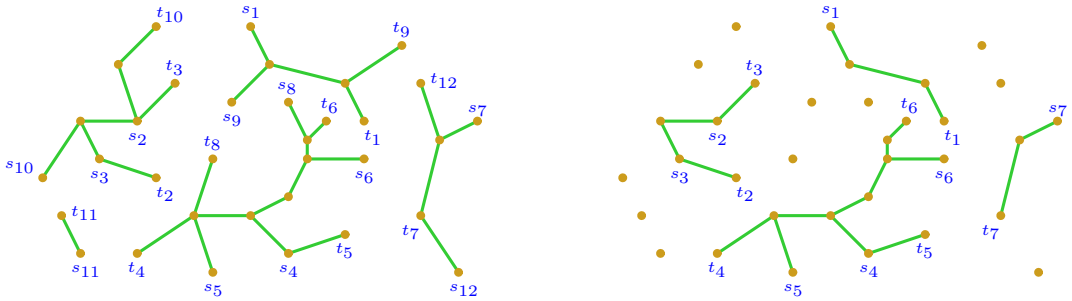


Figure 2.2: (Left) The optimal solution for an instance with $n = 11$. (Right) The connected components of the solution restricted to R for $k = 7$.

solution to the original unit demands problem, which we denote by OPT. Let e be an edge in the optimal solution with x_e units of demand routed along it. This means that there are x_e single unit demand source-sink pairs that utilize e. The probability that at least one of those pairs belongs to R is at most $\min\left\{1, x_e\frac{k}{n}\right\}$. The cost of routing $\frac{n}{k}$ units of demand along e is at most $f_e\left(\frac{n}{k}\right)$ no matter whether there already is some demand routed along e or not (f_e is sub-additive). Let c_e^R denote the cost of routing $\frac{n}{k}$ additional units of demand along e. Hence $\mathrm{E}_\pi[c_e^R] \leq \min\left\{1, x_e\frac{k}{n}\right\} f_e\left(\frac{n}{k}\right)$. It is easy to see that $\mathrm{E}_\pi[c_e^R] \leq f_e(x_e)$:

- If $x_e\frac{k}{n} > 1$, then $\mathrm{E}_\pi[c_e^R] \leq f_e\left(\frac{n}{k}\right) \leq f_e(x_e)$ since f_e is nondecreasing.
- If $x_e\frac{k}{n} \leq 1$, then $\mathrm{E}_\pi[c_e^R] \leq x_e\frac{k}{n} f_e\left(\frac{n}{k}\right) = x_e\frac{f_e\left(\frac{n}{k}\right)}{\frac{n}{k}} \leq f_e(x_e)$ since f_e is sub-additive.

The above argument shows that $\mathrm{E}_\pi\left[\sum_i OPT_i^R\right]$ is at most OPT. When constructing an alternating path connecting s_k to t_k, the segments that do not correspond to hops will consist of edges from the connected components $A_1, A_2, \ldots, A_l$. The cost for this portion of the alternating path will be exactly the cost of routing $\frac{n}{k}$ units of demand along those edges. Our goal is to show that we would need to pay only a small fraction of $\mathrm{E}_\pi\left[\sum_i OPT_i^R\right]$.

Now consider one of the connected components A_i. Construct a tour P_i that passes through all source-sink pairs and utilizes each edge no more than twice. If A_i is a tree, double all its edges to obtain an Eulerian tour P_i. Otherwise, first remove edges from A_i to break up the cycles and then double the remaining edges to obtain the tour P_i. The cost of routing $\frac{n}{k}$ units of demand along the edges of P_i is bounded by $2\,OPT_i^R$.

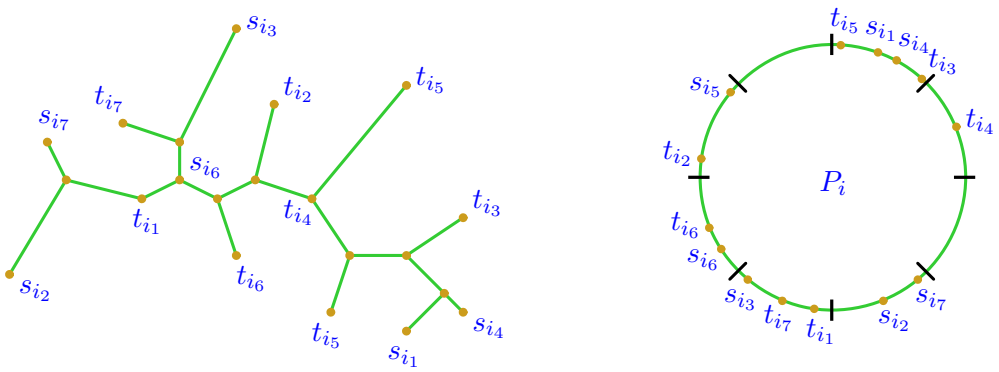


Figure 2.3: (Left) Connected component A_i containing pairs $(s_{i_1}, t_{i_1}), (s_{i_2}, t_{i_2}), \ldots, (s_{i_7}, t_{i_7})$. (Right) The corresponding Eulerian tour P_i, divided into segments.

We will construct an auxiliary graph corresponding to P_i such that cycles in the auxiliary graph that involve the pair (s_k, t_k) would correspond to augmenting paths connecting s_k to t_k. Bounding the girth of the auxiliary graph will give us a bound on the number of hops in an augmenting path connecting (s_k, t_k).

Auxiliary graph construction

If $k_i \geq e^{\sqrt{\ln k \ln\ln k}}$, divide P_i into $n_i = \lfloor k_i^{1-2/g_k} \rfloor$ segments of cost at most $2\,OPT_i^R/n_i$. This is easily achieved by picking an arbitrary point along P_i and traversing the cycle in one direction, breaking up edges where the cost of the current segment exceeds the average segment cost. We will set $g_k > 2$, which is sufficient to guarantee that $n_i \geq 2$ for k large enough.

Now construct an auxiliary graph H_1^i with n_i vertices corresponding to the n_i segments of P_i. Two vertices u and v are connected by an edge in H_1^i if and only if there is a source-sink pair from R with a source in one of the segments corresponding to u and v and a sink in the other such segment (see Figure 2.4). Notice that we are allowing multiple edges and self loops in H_1^i. The resulting graph H_1^i has $n_i = \lfloor k_i^{1-2/g_k} \rfloor$ vertices and k_i edges. We will use the following result, due to Alon et al. [4], to bound the girth of H_1^i:

Theorem 2.3.3 ([4]) *The number of vertices n in a graph of girth g and average degree*

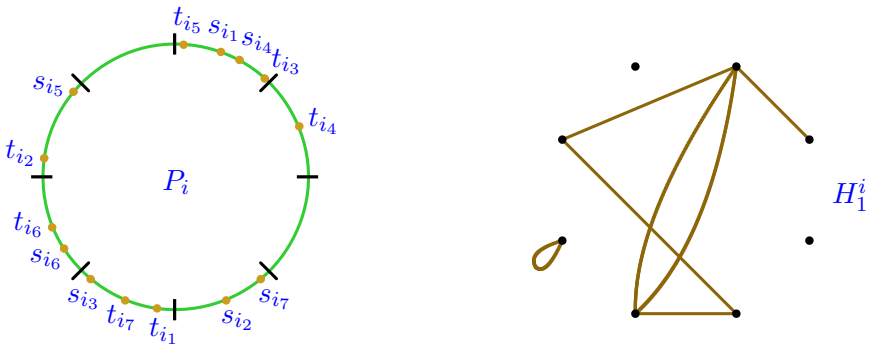


Figure 2.4: (Left) The Eulerian cycle P_i. (Right) The corresponding auxiliary graph H_1^i.

at least $d \geq 2$, *satisfies:*

$$n \geq 4 \left\lfloor \frac{d}{2} \right\rfloor^{\frac{g-2}{2}}$$

Lemma 2.3.4 *For* $g_k < 4\sqrt{\frac{\ln k}{\ln \ln k}}$ *and* k *large enough, the girth of* H_1^i *is less than* g_k.

Proof: The average degree of H_1^i is $\frac{2k_i}{n_i} = \frac{2k_i}{\lfloor k_i^{1-2/g_k} \rfloor} \geq 2$ and the condition of Theorem 2.3.3 is satisfied. If g denotes the girth of H_1^i then

$$\begin{aligned} g &\leq 2\,\frac{\ln n_i - \ln 4}{\ln \lfloor \frac{k_i}{n_i} \rfloor} + 2 \\ &\leq 2\,\frac{\ln k_i^{1-2/g_k} - \ln 4}{\ln \lfloor k_i^{2/g_k} \rfloor} + 2 \\ &= \frac{(g_k - 2)\,\ln k_i^{2/g_k} - 2\ln 4}{\ln \lfloor k_i^{2/g_k} \rfloor} + 2 \\ &< \frac{(g_k - 2)\,\ln \left(k_i^{2/g_k} - 1\right)}{\ln \lfloor k_i^{2/g_k} \rfloor} + 2 \\ &\leq g_k \end{aligned}$$

In the second to last inequality we use the fact that $k_i^{2/g_k} \geq g_k$ for the choice of k_i, g_k and k large enough. This allows us to complete the proof by showing that

$$\ln \frac{k_i^{2/g_k}}{k_i^{2/g_k} - 1} \leq \frac{1}{k_i^{2/g_k} - 1} < \frac{2\ln 4}{g_k - 2}$$

□

Having the girth of H_1^i less than g_k means that there exists a cycle B_1^i in H_1^i of length $b_1^i < g_k$. The cost of routing $\frac{n}{k}$ units of demand along the segments of P_i corresponding to the vertices in B_1^i is at most

$$2OPT_i^R \lfloor k_i^{1-2/g_k} \rfloor^{-1} b_1^i < 4\, k_i^{2/g_k - 1} b_1^i\ OPT_i^R$$

Cycle decomposition of P_i

Recall that edges of the cycle B_1^i correspond to source-sink pairs in R. We would like to partition all source-sink pairs into low-length cycles like B_1^i. Think about removing from consideration the source-sink pairs that made up the edges of B_1^i. If the number of remaining source-sink pairs is still large enough (i.e. at least $e^{\sqrt{\ln k \ln\ln k}}$) we repeat the auxiliary graph argument to construct a graph H_2^i by splitting P_i into equal cost segments. The number of vertices of H_2^i is $\lfloor (k_i - b_1^i)^{1-2/g_k} \rfloor$ and the number of edges is $k_i - b_1^i$. Then there exists a cycle B_2^i in H_2^i of length $b_2^i < g_k$ and the cost of routing $\frac{n}{k}$ units of demand along the segments of P_i corresponding to the vertices in B_2^i is at most $4(k_i - b_1^i)^{2/g_k - 1} b_1^i\ OPT_i^R$.

We continue this procedure as long as the number of remaining source-sink pairs in A_i that have not been associated with some cycle B_j^i is at least $e^{\sqrt{\ln k \ln\ln k}}$. We end up with a collection of cycles $B_1^i, \ldots, B_p^i$ with lengths $b_j^i < g_k$ such that $\sum_{j=1}^{p} b_j^i = k_i' \leq k_i$. We also have $k_i - k_i' < e^{\sqrt{\ln k \ln\ln k}}$ other source-sink pairs that do not belong to any cycle B_j^i. If the pair (s_k, t_k) belongs to a cycle, we will use the alternating path corresponding to that cycle to route the required $\frac{n}{k}$ units of demand. We will start at s_k, traverse the segment it is in to find the terminal of the source-sink pair that corresponds to the next edge in the cycle, then take a hop to the other terminal of that pair and continue in this manner until we reach t_k. The cost of the optimal solution segments of such a path will be bounded by the cost of the cycle. If the pair (s_k, t_k) does not belong to any cycle we will route its demand along an alternating path with no hops that simply traverses P_i to connect s_k to t_k.

The cost of routing $\frac{n}{k}$ units of demand along the segments corresponding to the vertices of a cycle B_j^i is at most $4(k_i - b_1^i - \ldots - b_{j-1}^i)^{2/g_k - 1} b_j^i\ OPT_i^R$. The bound on the cost of routing $\frac{n}{k}$ units of demand along the vertices of any given cycle might be quite large.

However, since R is a random selection of k pairs and each of those pairs is equally likely to have been chosen as the k-th one, the expected cost of servicing (s_k, t_k) will be averaged across all cycles. The probability that we use the cycle B_j^i as the one to route demand along, given that we use A_i, is $\frac{b_j^i}{k_i}$. Then the expected cost of routing $\frac{n}{k}$ units of demand along the segments corresponding to the vertices in the cycle to which (s_k, t_k) belongs, given that it is in A_i, is less than

$$\sum_{j=1}^{p} 4(k_i - b_1^i - \ldots - b_{j-1}^i)^{2/g_k - 1} b_j^i \, \mathrm{E}_\pi \left[OPT_i^R \right] \, \frac{b_j^i}{k_i}$$

With probability $\frac{k_i - k_i'}{k_i}$ the pair (s_k, t_k) will not belong to any of the cycles B_j^i and the expected cost of routing $\frac{n}{k}$ units of demand along optimal solution edges between s_k and t_k given that (s_k, t_k) is in A_i will be $\mathrm{E}_\pi \left[OPT_i^R \right] \, \frac{k_i - k_i'}{k_i}$. Therefore the expected cost of routing $\frac{n}{k}$ units of demand along the optimal solution edges of an alternating path connecting s_k to t_k as described by an auxiliary graph cycle or a direct path with no hops, given that (s_k, t_k) is in A_i is less than

$$\sum_{j=1}^{p} 4(k_i - b_1^i - \ldots - b_{j-1}^i)^{2/g_k - 1} b_j^i \, \mathrm{E}_\pi \left[OPT_i^R \right] \, \frac{b_j^i}{k_i} + \mathrm{E}_\pi \left[OPT_i^R \right] \, \frac{k_i - k_i'}{k_i}$$

In order to simplify this expression we will use the following:

Lemma 2.3.5 *For* $1 \leq b_j^i < g_k$ *and* $g_k > 2$

$$\sum_{j=1}^{p} 4(k_i - b_1^i - \ldots - b_{j-1}^i)^{2/g_k - 1} {b_j^i}^2 \leq 2 k_i^{2/g_k} g_k^2$$

Proof: Introduce new variables $a_j \in (0, 1)$ for $j = 1, 2, \ldots, k_i$ such that $a_j = b_j^i / g_k$ for $j = 1, 2, \ldots, k_i'$ and $a_j = 1/g_k$ for $j = k_1' + 1, \ldots, k_i$. Then the left hand side of the inequality can be rewritten as

$$\begin{aligned}
& \sum_{j=1}^{p} 4(a_{p+k_i-k_i'} + \ldots + a_j)^{2/g_k - 1} {a_j}^2 g_k^{2/g_k + 1} \\
\leq & \sum_{j=1}^{p+k_i-k_i'} 4(a_{p+k_i-k_i'} + \ldots + a_j)^{2/g_k - 1} a_j g_k^{2/g_k + 1} \\
\leq & \; 4 g_k^{2/g_k + 1} \int_0^{k_i/g_k} t^{2/g_k - 1} \mathrm{d}t \\
\leq & \; 2 k_i^{2/g_k} g_k^2
\end{aligned}$$

□

The lemma above allows us to bound the optimal solution portion of the alternating path connecting s_k to t_k, given that the pair is in A_i by

$$\left(2k_i^{2/g_k-1}g_k^2 + \frac{1}{k_i}e^{\sqrt{\ln k \ln\ln k}}\right) \mathrm{E}_\pi\left[OPT_i^R\right]$$

The probability that the k-th pair is in component A_i is $\frac{k_i}{k}$. Hence the expected cost of routing $\frac{n}{k}$ units of demand along the optimal solution edges of an alternating path connecting s_k to t_k as described by an auxiliary graph cycle or a direct path with no hops is less than

$$\begin{aligned}
& \sum_{i=1}^{l}\left(2k_i^{2/g_k-1}g_k^2 + \frac{1}{k_i}e^{\sqrt{\ln k \ln\ln k}}\right)\ \frac{k_i}{k}\ \mathrm{E}_\pi\left[OPT_i^R\right] \\
= \quad & \frac{2g_k^2}{k}\sum_{i=1}^{l}k_i^{2/g_k}\ \mathrm{E}_\pi\left[OPT_i^R\right] + \frac{1}{k}e^{\sqrt{\ln k \ln\ln k}}\sum_{i=1}^{l}\mathrm{E}_\pi\left[OPT_i^R\right] \\
\leq \quad & 2g_k^2\ k^{2/g_k-1}\ OPT + \frac{1}{k}e^{\sqrt{\ln k \ln\ln k}}\ OPT
\end{aligned}$$

Let $(s_{i_1}, t_{i_1}), \ldots (s_{i_b}, t_{i_b}) \in R \setminus \{(s_k, t_k)\}$ be the pairs that are responsible for the edges of the cycle (s_k, t_k) belongs to, where $0 \leq b < g_k - 1$. Consider the alternating path that starts at s_k, reaches s_{i_1} along edges of P_i, hops to t_{i_1}, reaches s_{i_2} along edges of P_i, hops to t_{i_2}, etc. until it does one last hop to t_{i_b} and from there finally reaches t_k along edges of P_i (see Figure 2.5). This is an alternating path with less than $g_k - 1$ hops that connects s_k to t_k and therefore the expected value of C_k^R is at most equal to the expected cost of this path. The result above places an upper bound on the expected value of the α_k^R portion of C_k^R for this path:

$$\mathrm{E}_\pi[\alpha_k^R] \leq 2g_k^2\ k^{2/g_k-1}OPT + \frac{1}{k}e^{\sqrt{\ln k \ln\ln k}}\ OPT$$

2.3.4 Bounding $\mathrm{E}_\pi\left[\beta_k^R\right]$

In this section we derive an upper bound on the cost of routing additional $\frac{n}{k}$ units of demand along the hops of an alternating path of the structure described in the previous section. This completes the upper bound on the cost of using an alternating path of that kind to route the inflated demand between s_k and t_k.

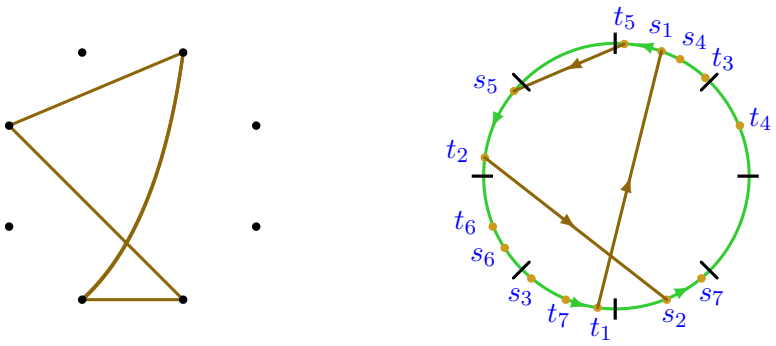


Figure 2.5: (Left) A cycle in the auxiliary graph. (Right) The augmenting path connecting (s_7, t_7) that corresponds to the cycle, $b = 3$.

For a given set R of the first k source-sink pairs, and pair $(s,t) \in R$, let $C_{s,t}^{R,i}$ denote the cost to connect pair (s,t) given that R is the set of the first k source-sink pairs and (s,t) was the i-th pair in the permutation. Notice that according to this notation $C_{s_k,t_k}^{R,k} = C_k^R$ and $\mathrm{E}_\pi[C_{s,t}^{R,i}] = C_i$.

Consider the decomposition into cycles (with edges corresponding to pairs in R and length less than g_k) constructed by our analysis in the previous section. The probability that the hop corresponding to the pair (s,t) is used to route the k-th pair is precisely the probability that some pair in the cycle involving (s,t) is the k-th pair, i.e. at most $\frac{g_k}{k}$. Further, if (s,t) was the i-th pair, a demand of $\frac{n}{i}$ was routed from s to t. The cost of routing additional $\frac{n}{k}$ units of demand is at most $\frac{i}{k} C_{s,t}^{R,i}$ (the edge cost functions are sub-additive). Since we pick a random permutation π, the pair (s,t) is equally likely to be any of the first k pairs. Hence, the expected cost of routing additional $\frac{n}{k}$ units of demand along the hop corresponding to (s,t) is at most

$$\frac{1}{k-1} \sum_{i=1}^{k-1} \frac{i}{k} \mathrm{E}_\pi[C_{s,t}^{R,i}] = \frac{1}{k-1} \sum_{i=1}^{k-1} \frac{iC_i}{k}$$

This bounds the expected contribution of pair (s,t) to $\mathrm{E}_\pi[\beta_k^R]$ by

$$\frac{g_k}{k} \frac{1}{k-1} \sum_{i=1}^{k-1} \frac{iC_i}{k}$$

Then

$$\begin{aligned} \mathrm{E}_\pi[\beta_k^R] &\leq \sum_{(s,t)\in R} \frac{g_k}{k} \frac{1}{k-1} \sum_{i=1}^{k-1} \frac{iC_i}{k} \\ &\leq \frac{g_k}{k-1} \sum_{i=1}^{k-1} \frac{i\frac{1}{k}\sum_{(s,t)\in R} C_i}{k} \\ &= \frac{g_k}{k-1} \sum_{i=1}^{k-1} \frac{iC_i}{k} \end{aligned}$$

2.3.5 Refinement and Analysis of the C_k Bound

As $C_k^R = \alpha_k^R + \beta_k^R$, sections 2.3.3 and 2.3.4 give us a bound on $C_k = \mathrm{E}_\pi[C_k^R]$ for $k \geq N_c$ where N_c is a constant. For $k < N_c$ we use the trivial bound on C_k that states that $C_k \leq OPT$. This is summarized below:

Theorem 2.3.6 *For* $2 < g_k < 4\sqrt{\frac{\ln k}{\ln\ln k}}$

$$C_k \leq \begin{cases} OPT & \text{for } k < N_c \\ 2g_k^2 k^{2/g_k - 1} OPT + \frac{1}{k} e^{\sqrt{\ln k \ln\ln k}}\ OPT + g_k \frac{\sum_{i=1}^{k-1} \frac{i}{k} C_i}{k-1} & \text{for } k \geq N_c \end{cases}$$

Notice that if we try to solve the recurrence relation above as it is, it would be difficult to obtain a very good upper bound on C_k since the contribution of the summands $\frac{i}{k}C_i$ from the β_k^R part of the bound would be almost equal to C_i as i gets close to $k-1$. Intuitively, the first pairs in the summation are the ones who overpay by a lot as they pay for routing $\frac{n}{1}, \frac{n}{2}, \ldots$ units of demand. This means that if we use the paths corresponding to some fraction of the first pairs only, the per unit routing cost would be small for adding $\frac{n}{k}$ units of demand to those paths. Following this reasoning, we would like to use only hops corresponding to the first $\left\lfloor \frac{k}{r_k} \right\rfloor$ pairs out of the $k-1$ that are already connected. r_k is a parameter whose value will be specified later such that $\frac{k}{k-1} \leq r_k \leq k$. Reasoning analogous to the analysis in sections 2.3.3 and 2.3.4 yields

Theorem 2.3.7 *For* $2 < g_k < 4\sqrt{\frac{\ln \lfloor \frac{k}{r_k} \rfloor}{\ln \ln \lfloor \frac{k}{r_k} \rfloor}}$, r_k *such that* $\frac{k}{k-1} \leq r_k \leq k$ *and* N'_c *a constant*

$$C_k \leq \begin{cases} OPT & \text{for } k < N'_c \\ 2g_k^2 \left\lfloor \frac{k}{r_k} \right\rfloor^{2/g_k - 1} OPT + \frac{1}{\lfloor \frac{k}{r_k} \rfloor} e^{\sqrt{\ln \lfloor \frac{k}{r_k} \rfloor \ln \ln \lfloor \frac{k}{r_k} \rfloor}} OPT + g_k \frac{\sum_{i=1}^{\lfloor \frac{k}{r_k} \rfloor - 1} \frac{i}{k} C_i}{\left\lfloor \frac{k}{r_k} \right\rfloor - 1} & \text{for } k \geq N'_c \end{cases}$$

Theorem 2.3.7 gives us a family of recursive bounds on C_k. We need to select the values for g_k and r_k that would allow us to solve the recurrence relation and obtain a good bound on $\mathrm{E}_\pi[T_n] = \sum_{k=1}^{n} C_k$, which is itself an upper bound on the performance of Inflated Greedy. The following lemma shows a (near optimal) choice for the parameters g_k and r_k.

Lemma 2.3.8 *The choice of* $\gamma = \frac{1+\sqrt{5}}{2} + \epsilon$, *where* $\epsilon > 0$, $r_k = e^{\frac{\gamma - 1 + 1/\gamma}{2} \sqrt{\ln k \ln \ln k}}$, *and* $g_k = 2 \left\lceil \sqrt{\frac{\ln k}{\ln \ln k}} \right\rceil$ *satisfies the requirements of Theorem 2.3.7 for* k *large enough.*

The above choice of parameters g_k and r_k allows us to prove that

Lemma 2.3.9

$$C_k \leq \begin{cases} OPT & \text{for } k < N'_c \\ \frac{N'_c}{k} e^{\gamma \sqrt{\ln k \ln \ln k}} OPT & \text{for } k \geq N'_c \end{cases}$$

Proof: We proceed by induction on k. The first inequality holds trivially for any $k < N'_c$. Suppose that the second inequality holds for all i such that $N'_c \leq i \leq k-1$. Then

$$\begin{aligned} k\, C_k &< 2\, k\, g_k^2 \left\lfloor \frac{k}{r_k} \right\rfloor^{2/g_k - 1} OPT + \frac{k}{\lfloor \frac{k}{r_k} \rfloor} e^{\sqrt{\ln \lfloor \frac{k}{r_k} \rfloor \ln \ln \lfloor \frac{k}{r_k} \rfloor}} OPT + g_k \frac{\sum_{i=1}^{\lfloor \frac{k}{r_k} \rfloor - 1} i\, C_i}{\left\lfloor \frac{k}{r_k} \right\rfloor - 1} \\ &< 2\, k\, g_k^2 \left(\frac{k}{2 r_k} \right)^{2/g_k - 1} OPT + 2\, r_k\, e^{\sqrt{\ln \frac{k}{r_k} \ln \ln \frac{k}{r_k}}} OPT \\ &\quad + g_k \frac{N'_c (N'_c - 1)}{\frac{k}{r_k}} OPT + g_k\, N'_c\, e^{\gamma \sqrt{\ln \frac{k}{r_k} \ln \ln \frac{k}{r_k}}} OPT \\ &< 64\, k \frac{\ln k}{\ln \ln k} \left(\frac{k}{r_k} \right)^{\sqrt{\frac{\ln \ln k}{\ln k}} - 1} OPT + 2\, r_k\, e^{\sqrt{\ln k \ln \ln k}} OPT \\ &\quad + g_k\, r_k\, N'_c\, OPT + 4 \sqrt{\frac{\ln k}{\ln \ln k}}\, e^{\gamma \sqrt{(\ln k - \ln r_k) \ln \ln k}} N'_c\, OPT \end{aligned}$$

In the above derivations, we use various floor/ceiling inequalities: if $x \geq 3$ then $\lfloor x \rfloor^\alpha < (x/2)^\alpha$ for $\alpha < 0$ and $\lfloor x \rfloor - 1 \geq x/2$; if $x \geq 1$, $\lceil x \rceil \leq 2x$. It is now easy to check that for $r_k = e^{\frac{\gamma - 1 + 1/\gamma}{2}\sqrt{\ln k \ln\ln k}}$ each of the four final terms can be bounded by $\frac{N'_c}{4} e^{\gamma\sqrt{\ln k \ln\ln k}} OPT$ for k larger than some constant independent of N'_c (consequently we can make sure we have N'_c greater than this constant). We omit the tedious but routine algebraic manipulations.

□

Finally, we need to compute the expected value of the total cost of Inflated Greedy.

Lemma 2.3.10 *For C_k as given in Lemma 2.3.9, γ as given in Lemma 2.3.8,*

$$\mathrm{E}_\pi[T_n] \leq \sum_{k=1}^{n} C_k < N'_c\ OPT + N'_c\ \ln n\ e^{\gamma\sqrt{\ln n \ln\ln n}} OPT$$

Proof: Using the results from lemma 2.3.9, we have

$$\begin{aligned}
\sum_{k=1}^{n} C_k &< N'_c\ OPT + \sum_{k=N'_c}^{n} \frac{N'_c}{k}\ e^{\gamma\sqrt{\ln k \ln\ln k}} OPT \\
&\leq N'_c\ OPT + N'_c\ e^{\gamma\sqrt{\ln n \ln\ln n}}\ OPT \sum_{k=N'_c}^{n} \frac{1}{k} \\
&< N'_c\ OPT + N'_c\ \ln n\ e^{\gamma\sqrt{\ln n \ln\ln n}}\ OPT
\end{aligned}$$

□

Thus we have proved the following theorem.

Theorem 2.3.11 Inflated Greedy *approximates the non-uniform multi-source multi-commodity buy-at-bulk network design problem up to a factor of $e^{O(\sqrt{\log n \log\log n})}$, where n is the total number of source-sink pairs and all source-sink pairs have unit demands.*

2.4 Inflated Greedy in the Single-Source Case

It is interesting to note that the performance of Inflated Greedy in the single-source case is far lower than its guarantee of $e^{O(\sqrt{\log n \log\log n})}$ for the general case. In this section

we will show that the algorithm gives an approximation of $O(\log^2 n)$ in the single-source case. This guarantee is not too far from the best known guarantee of $O(\log n)$ by [59].

The analysis proceeds in a similar manner to the one of the multi-source version detailed in the previous section. A lot of the intricate details of the analysis are identical, hence only the most salient features particular to the single-source case are presented.

Notice that since we are in the single-source case, $s_1 = s_2 = \ldots = s$. We define α_k^R and β_k^R as before: α_k^R is the cost of routing $\frac{n}{k}$ units of demand along edges of the optimal solution connecting already existing paths chosen by Inflated Greedy, given that R is the set of the first k source-sink pairs. β_k^R is the cost of routing additional $\frac{n}{k}$ units of demand along the already constructed paths, again given that R is the set of the first k source-sink pairs.

The optimal solution in the single-source case is a tree, so instead of looking at alternating paths with hops, we will concentrate only on paths that consist of a sequence of optimal solution edges used to connect t_k to some other terminal t_i, followed by the path chosen by Inflated Greedy to connect t_i to s. This greatly simplifies our analysis because we do not have to go through the auxiliary graph and girth argument as before.

2.4.1 Bounding $\mathrm{E}_\pi\left[\alpha_k^R\right]$

In Section 2.3 we proved that the expected cost of routing $\frac{n}{k}$ units of demand along the edges of the optimal solution is at most OPT. Let A denote the union of the paths used to connect the first k pairs in the optimal solution. As before, we double the edges of A to obtain an Eulerian tour P of expected cost at most $2OPT$. We connect t_k to the closest one of $s, t_1, \ldots, t_{k-1}$ along the cycle. Since π is a random permutation, the expected cost of connecting t_k will be equal to the average cost between any two of $s, t_1, \ldots, t_k$. This gives us

$$\mathrm{E}_\pi[\alpha_k^R] \leq \frac{2OPT}{k+1} < \frac{2OPT}{k}$$

2.4.2 Bounding $\mathrm{E}_\pi\left[\beta_k^R\right]$

Notice that since t_k connects to the closest one of $s, t_1, \ldots, t_{k-1}$ we would be using at most one of the existing paths. The expected cost of routing additional $\frac{n}{k}$ units of demand along the path used by the pair (s_i, t_i) is no greater than $\frac{i}{k}C_i$ and we are equally likely to have any of the first $k-1$ pairs as the closest one. Therefore we have

$$\mathrm{E}_\pi[\beta_k^R] \leq \frac{\sum_{i=1}^{k-1} \frac{i}{k} C_i}{k-1}$$

2.4.3 Analysis of the C_k Bound

The previous two subsections show that for $k > 1$

$$C_k < \frac{2OPT}{k} + \frac{\sum_{i=1}^{k-1} \frac{i}{k} C_i}{k-1}$$

Let $U_k = \sum_{i=1}^{k} i\, C_i$. Then

$$\begin{aligned} & U_k - U_{k-1} < 2OPT + \frac{U_{k-1}}{k-1} \\ \Leftrightarrow \quad & \frac{U_k}{k} < \frac{2OPT}{k} + \frac{U_{k-1}}{k-1} \end{aligned}$$

It immediately follows that $U_k < 2\,k\,H(k)\,OPT$ where $H(k) = 1 + \frac{1}{2} + \ldots + \frac{1}{k}$. Then

$$\begin{aligned} \mathrm{E}_\pi[T_n] \leq \sum_{k=1}^{n} C_k &= OPT + \sum_{k=2}^{n} \frac{U_k - U_{k-1}}{k} = \\ &= OPT + \frac{U_n}{n} - U_1 + \sum_{k=2}^{n-1} \frac{U_k}{k(k+1)} \\ &\leq OPT \left(1 + \frac{2\,\ln n}{n} + 2\,\ln^2 n\right) \end{aligned}$$

Thus we get an approximation of $O\left(\log^2 n\right)$ where n is the number of source-sink pairs.

Theorem 2.4.1 Inflated Greedy *approximates the non-uniform single-source multi-commodity buy-at-bulk network design problem up to a factor of* $O(\log^2 n)$*, where* n *is the number of source-sink pairs and all source-sink pairs have unit demands.*

2.5 Applying Inflated Greedy to the General Demands Case

The algorithm we presented and analyzed in the previous sections is designed to work in the case when all source-sink pairs have unit demands. This suggests that if we want to use Inflated Greedy in the general demands case we will need to apply it as a subroutine on a subset of the source-sink pairs that have almost the same demand.

We start with a simple observation about the optimal solution in the general demands case of the problem:

Lemma 2.5.1 *In the optimal solution to the general demands problem, the demand of each pair (s,t) is routed along a single path from s to t.*

Proof: Let the number of units routed along each edge e in the optimal solution be x_e^*. Suppose that the demand of pair (s,t) is routed along at least two paths P_1 and P_2. Since we are looking at the optimal solution, it would be more expensive to take away one unit of demand from the path P_1 and route it along the path P_2 and vice versa. This means that

$$\begin{aligned}\sum_{e\in P_2}(f_e(x_e^*+1)-f_e(x_e^*)) &> \sum_{e\in P_1}(f_e(x_e^*)-f_e(x_e^*-1))\\ \sum_{e\in P_1}(f_e(x_e^*+1)-f_e(x_e^*)) &> \sum_{e\in P_2}(f_e(x_e^*)-f_e(x_e^*-1))\end{aligned}$$

Adding the two inequalities and rearranging gives us

$$\sum_{e\in P_1}(f_e(x_e^*-1)+f_e(x_e^*+1)-2\,f_e(x_e^*))+\sum_{e\in P_2}(f_e(x_e^*-1)+f_e(x_e^*+1)-2\,f_e(x_e^*))>0$$

However, we know that the edge cost functions are sub-additive, so for any edge $e\in E$, $2\,f_e(x_e^*)\geq f_e(x_e^*-1)+f_e(x_e^*+1)$. This results in a contradiction and completes the proof. □

Consider an instance of the non-uniform multicommodity buy-at-bulk network design problem with a set of source-sink pairs $\mathcal{D}$ and demand function $\delta:\mathcal{D}\to\mathbb{Z}^+$. Let D be the value of the maximum demand of any source-sink pair, i.e. $D=\max\{\delta(s,t)\mid(s,t)\in\mathcal{D}\}$. Partition $\mathcal{D}$ into subsets $\mathcal{D}_1,\mathcal{D}_2,\ldots,\mathcal{D}_l$ such that $(s,t)\in\mathcal{D}_i$ if and only if $2^{i-1}\leq\delta(s,t)<2^i$.

The pair with the largest demand D is in $\mathcal{D}_l$ so $\log D < l \leq \log D + 1$. We modify the demands of all source-sink pairs in $\mathcal{D}_i$ to be 2^i thus inflating them by no more than a factor of 2. If the value of the optimal solution to the original instance is OPT and the value of the optimal solution to the modified instance is OPT', then $OPT' \leq 2\,OPT$ since all edge cost funcitons are sub-additive. Moreover, if the value of the optimal solution of the subproblem defined by the source-sink pairs in $\mathcal{D}_i$ is OPT'_i, then clearly $OPT'_i \leq OPT'$.

We are now ready to deal with each group $\mathcal{D}_i$ separately. Consider the subproblem defined on the original graph G with source-sink pairs in $\mathcal{D}_i$ and demands inflated as above. The optimal solution routes the demand of each source-sink pair along a single path between the terminals of the pair. Since all source-sink pairs in $\mathcal{D}_i$ have the same demand 2^i, this allows us to think of the 2^i units of original demand as a single "unit of demand". To that effect, for each edge e define a new cost function $f_e^i : \mathbb{Z}^+ \to \mathbb{R}^+$ such that $f_e^i(x) = f_e(2^i\,x)$ and set the demands to all pairs in $\mathcal{D}_i$ to be unit demands. The optimal solution to this new rescaled problem is exactly OPT'_i. Running Inflated Greedy on the newly created unit demands instance produces a solution of cost no greater than $e^{O(\sqrt{\log n \log\log n})} OPT'_i$.

Finally, we combine the solutions found when dealing with each of the demand groups $\mathcal{D}_i$. Since the edge cost functions are sub-additive, the cost of the combined solution is no greater than

$$\sum_{i=1}^{l} e^{O(\sqrt{\log n \log\log n})} OPT'_i \leq e^{O(\sqrt{\log n \log\log n})} \log D\ OPT$$

This completes the proof that the algorithm for the general demands case described above has an approximation factor of $e^{O(\sqrt{\log n \log\log n})} \log D$.

Chapter 3

Conclusion

We give the first non-trivial approximation algorithm for the non-uniform multicommodity buy-at-bulk network design problem. In an instance of the problem with unit demands our Inflated Greedy algorithm has an approximation factor guarantee of $e^{O(\sqrt{\log n \log\log n})}$. Inflated Greedy can be used as a subroutine in a very intuitive way to obtain a $e^{O(\sqrt{\log n \log\log n})} \log D$ approximation guarantee for the problem with general demands. A very attractive feature of our algorithm is its simplicity - it is essentially a randomized greedy algorithm and the intricacy lies within the analysis. Both Inflated Greedy and the algorithm for the general demands case only perform n shortest path calculations between two vertices in the graph, resulting in a running time of $O(n(|E| + |V| \log |V|))$.

One of the most interesting open problems was whether the approximation factor could be reduced to *polylog*(n), a possibility not ruled out by the best known hardness results at the time. Another open problem was to remove the dependance of the approximation factor on the largest demand in the graph. Both of these open problems were resolved by Chekuri et al. [22], who give an approximation algorithm with guarantee $O\left(\log^3 n \min\{\log D, \gamma(n^2)\}\right)$ where $\gamma(n)$ is the worst case distortion when embedding the metric induced by a graph on n vertices into a distribution over its spanning trees. Using the best known bound for $\gamma(n)$ ($O(\log^2 n \log\log n)$ by Elkin et al. [32]), this gives an $O\left(\min\{\log^3 n \log D, \log^5 n \log\log n\}\right)$ guarantee.

The algorithm proceeds iteratively. In each iteration it constructs a partial solution of low density that connects some of the remaining source-sink pairs and removes them from the graph. The density of the partial solution is defined as the ratio of the total cost of the partial solution to the number of pairs that were still unconnected at the beginning of the iteration. Using standard set-cover type analysis results in a polylogarithmic approximation guarantee.

The main idea behind constructing the low density partial solution is to show the existence of one with a specific structure (junction-tree). Chekuri et al. define a junction-tree with root r for a set of source-sink pairs A to be a tree that contains all source and sink terminals such that the unique path in the tree connecting the source and sink of each pair in A passes through the root r. They produce two approximation algorithms for finding a junction-tree of low density. For arbitrary demands, they use an LP relaxation for the single-source buy-at-bulk problem [23] to get an approximately low density junction-tree. When the total demand D is polynomial in n they give a greedy combinatorial algorithm that mainly uses a result on computing shallow-light trees by Hajiaghayi et al. [45] Putting everything together results in an approximation algorithm for the general buy-at-bulk problem with the aforementioned polylogarithmic approximation guarantee.

The algorithms produce the above approximation guarantee that is a notable improvement over the performance guarantee of our algorithm, yet the final algorithm is conceptually harder to understand and more complicated. It would be interesting to design a conceptually simpler algorithm with similar or better approximation guarantee.

There is still a gap between the best known hardness result for the non-uniform buy-at-bulk network design problem ($O(\log^{1/2} n)$ by Andrews [5]) and the performance of the best algorithm (at best $O(\log^3 n)$). Closing this gap is the next interesting open problem.

Another avenue to explore is algorithms for the online non-uniform buy-at-bulk problem, where the source-sink pairs are presented one at a time and the goal is to obtain a low competitive ratio, i.e. produce a solution of cost not too much greater than the optimal solution had all the source-sink pairs been known in advance. To the best of our knowledge,

the only version of the buy-at-bulk problem that has been successfully attacked in the online setting is the single-source uniform case, under the name of *access network design* problem, where a there is a constant factor approximation algorithm.

Part II

Terminal Backup, 3D Matching and Covering Cubic Graphs

Chapter 4

Introduction

We study a problem from matching theory that we call *Simplex Matching*, originally motivated by looking at the *Terminal Backup* network design problem. In this chapter we introduce both problems and discuss how *Terminal Backup* can be solved in polynomial time if we had a polynomial time algorithm for *Simplex Matching*. We also briefly discuss related work. A polynomial time algorithm for *Simplex Matching* will be presented in the next chapter.

4.1 Motivation

Consider the problem of designing networks that can withstand the failure of some of the links that comprise them. In order to achieve such survivability, we need to install an appropriate number of redundant links while still keeping the overall cost of the network low. In some networks the redundancy can be planned at the same time as the design of the original network, while in others the extra links need to be added afterwards.

A specific problem that fits this framework is the problem of designing access networks that are a part of communication networks that connect subscribers to their service providers. The structure of such a network is a rooted tree of customers at the leaves and remote terminals at the intermediate nodes. If a link from a terminal to its parent fails, the terminal has to relay the traffic through another terminal of the same or higher level [74].

One of the parameters of the problem is whether the network between the remote terminals is a single level tree (star network) or a multi level tree. If the network is a multi level tree we have the *Survivable Multi-Level Fat Tree* problem [74], where we are given a graph G with terminal nodes, non-terminal nodes and edge weights. There is an existing rooted tree T spanning the terminals that does not contain non-terminal nodes. The goal is to buy the cheapest set of edges such that every terminal can connect to the root of T even after any single edge failure.

If the access tree between the remote terminals is a single level tree then we need to augment the network in such a way that each terminal is connected to either one other terminal or to the root. This can be modeled by the *Terminal Backup* problem:

Definition 4.1.1 (Terminal Backup) *Given a graph consisting of terminal nodes, non-terminal nodes and edges with costs associated to them, construct a network of minimum cost that connects each terminal to at least one other terminal.*

In *Terminal Backup*, the terminal nodes represent the facilities that need to be connected for backup purposes. To do this we construct a network where every facility is connected to at least one other facility. In other words, we need to find a forest of minimum cost such that each of its connected components contains at least two terminals.

In [74], Xu et al. define the *Survivable Multi-level Fat Tree* and *Terminal Backup* problems. They show that the former is NP-hard via a reduction from *Geometric Steiner Tree* and give a 2-approximation algorithm for the problem that has as a subroutine a polynomial time algorithm for *Terminal Backup*. Concurrently with [74] and motivated by the *Terminal Backup* problem they define, we developed a polynomial time algorithm for solving the *Simplex Matching* problem that can be used easily to solve *Terminal Backup* (joint work with E. Anshelevich [8]). This algorithm and its proof of correctness form the basis of the second part of this dissertation.

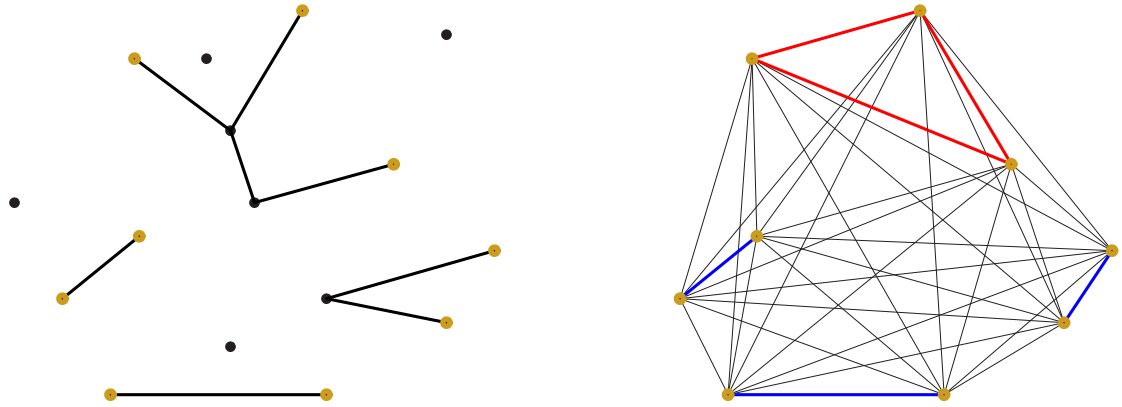


Figure 4.1: (Left) An instance of *Terminal Backup* with its optimal solution, orange nodes are terminals. (Right) The corresponding hypergraph with a cover that corresponds to the optimal solution to the left, blue edges are 2d edges, the red edges form a 3d edge.

4.2 Simplex Cover

In the optimal solution of the *Terminal Backup* problem the removal of an edge would disconnect at least one of the terminals. Therefore each connected component of the optimal solution must consist of terminals connected to a central vertex (that may or may not be a terminal) via a path that contains no terminal nodes. Moreover each such "star" can be broken down further into components that connect 2 or 3 terminals each.

This structure of the optimal solution suggests an equivalent formulation of the problem. Given a graph G with terminal nodes $\mathcal{T}$ and edge costs c_e, form a hypergraph $H(\mathcal{T}, E)$ with edges of size 2 and 3. For all terminals $x, y \in \mathcal{T}$ form a 2d edge in H of cost $c(x, y)$ equal to the cost of the shortest path between x and y in G. Similarly, for all triples of terminals $x, y, z \in \mathcal{T}$ form a 3d edge in H of cost $c(x, y, z)$ equal to the cost of the cheapest way to connect all three x, y and z in G. The optimal solution to the *Terminal Backup* instance defined by G and $\mathcal{T}$ corresponds to a cover of H by 2d and 3d edges (see Figure 4.1).

The problem of finding a minimum cost cover of the vertices of a hypergraph H with edges of size 2 and 3 is not exactly equivalent to *Terminal Backup*. An additional constraint on the edge costs (the *Simplex Condition*) is needed to complete the equivalence:

Definition 4.2.1 (Simplex Condition) *Given a hypergraph (H, E) with edges of size 2 and 3, if $(x, y, z) \in E$ then $(x, y), (y, z), (x, z) \in E$ and*

$$c(x, y) + c(y, z) + c(x, z) \leq 2 \cdot c(x, y, z)$$

It is easy to see that the hypergraph corresponding to an instance of *Terminal Backup* satisfies the *Simplex Condition*. First note that all 2d and 3d edges exist. Consider any three terminals $x, y, z \in \mathcal{T}$. Without loss of generality we can assume that the cheapest way (of cost $c(x, y, z)$) to connect them in the original graph G utilizes some non-terminal vertex p – if the cheapest way to connect them was a path between say x and z passing through y then we could always create an extra non-terminal p that is at a distance 0 from y and distance $c(y, v)$ to any other vertex v in G. If we only wanted to connect two of the terminals, we could not do worse than a path passing through p. This immediately shows that $c(x, y) + c(y, z) + c(x, z) \leq 2 \cdot c(x, y, z)$ (see Figure 4.2(b)).

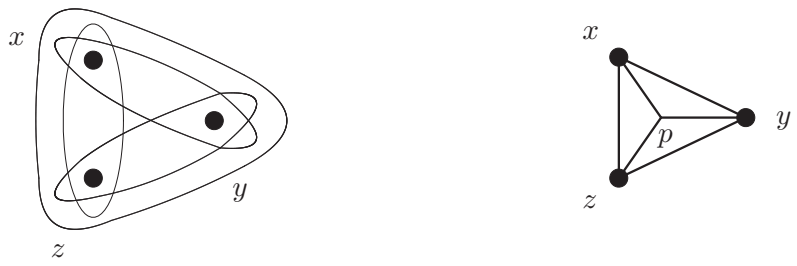


Figure 4.2: The Simplex Condition in *Simplex Matching* and *Terminal Backup*.

We can now define *Simplex Cover* and claim that it is equivalent to *Terminal Backup*:

Definition 4.2.2 (Simplex Cover) *Given a hypergraph with edges of size* 2 *and* 3 *that satisfies the* Simplex Condition*, find the minimum cost edge cover.*

Theorem 4.2.3 *([74]) The* Simplex Cover *problem is equivalent to* Terminal Backup.

4.3 Simplex Matching

A reduction similar to the one from perfect matching to edge cover (as in [68]) can show that *Simplex Cover* can be solved via its matching version that we call *Simplex Matching*. In a hypergraph, a perfect matching (often referred to as a packing) is simply a collection of edges of the hypergraph such that every node of the graph appears in exactly one of those edges.

Definition 4.3.1 (Simplex Matching) *Given a hypergraph with edges of size* 2 *and* 3 *that satisfies the* Simplex Condition*, find the minimum cost perfect matching.*

We will briefly outline the construction from [74] that is used in the reduction from *Simplex Cover* to *Simplex Matching*. Given an instance $H(\mathcal{T}, E)$ of *Simplex Cover* make another copy of H called H' and for every node x from H and its corresponding node x' from H' form an edge (x, x') with weight $2 \min_{y:(x,y)\in E} c(x, y)$. Solving *Simplex Matching* on this augmented graph results in a minimum cost perfect matching M consisting of edges in H, edges in H' and edges between vertices in H and H'. If a vertex x from H is not covered by an edge that goes between H and H' then its corresponding vertex x' from H' is also not covered by such an edge. Now it is easy to see that the cost of the selected edges in H should be the same as the cost of the selected edges in H' otherwise the perfect matching could be improved. To construct a feasible solution to *Simplex Cover* in H, first take the portion of M that is entirely in H. For all vertices $x \in E$ that were covered by (x, x') edges in M, add $(x, y) \in E$ with cost $c(x, y) = \frac{c(x,x')}{2}$ to cover x. It is not difficult to see that this feasible solution to *Simplex Cover* has in fact minimum cost (the proof of this fact can be found in Theorem 4 of [74]).

Note that *Simplex Matching* would be NP-hard without the additional constraints imposed by the *Simplex Condition*. This is easy to see by observing that if the hypergraph contained edges of size 3 only we have exactly 3d-matching. In the next chapter we show that *Simplex Matching* is solvable in polynomial time. This solves *Terminal Backup*, which was the original motivation of our work.

4.4 Related Work

Terminal Backup is similar to many Steiner-tree variations [20, 33, 44]. However, all such variations are required to either connect particular pairs of terminals, connect terminals from a particular set, or connect at least k terminals in total. The problem of finding the cheapest forest with at least k terminals in *each* component has not been addressed before. In addition, all of the above variations are NP-Hard while *Terminal Backup* is solvable in polynomial-time for $k = 2$. For $k > 2$ the problem becomes NP-Hard, although there is a 2-approximation algorithm using techniques from [37].

Matching theory, as well as its extensions, is both extremely important and well-studied. Perhaps surprisingly, there still remain basic matching problems that can be solved efficiently, and yet are not solvable using existing matching algorithms and techniques [26, 29, 46, 47]. We address one such problem, *Simplex Matching*, and show how to solve it in polynomial time using an elegant covering argument. While *Simplex Matching* is interesting in its own right as a nontrivial extension of non-bipartite minimum cost matching, its main value lies in many (seemingly very different) problems that can be solved using our algorithm. In the last chapter we will briefly discuss how *Simplex Matching* can be applied to the *Project Assignment* problem in which we have a set of projects and students with project preferences. The goal is to break up the students into groups of at least 2 and assign a project to each group so as to maximize the total student "happiness".

Simplex Matching and especially *Project Assignment* are also very similar to variants of facility location. In fact, we can use *Simplex Matching* to solve instances of facility location where all open facilities have lower bounds of 2 and the facility costs obey the *Simplex Condition* (the costs are all 0 or the cost of serving three clients is at least twice the cost of serving two clients). Although this is a very special case of facility location, it is the first result (to our knowledge) of a non-trivial facility location problem with lower bounds [2, 40, 51] that can be solved efficiently.

Matching theory is a very large field (see [55]) and there are many algorithms for weighted non-bipartite matching. A lot of work has also been done on exact *packings*,

which are exact covers of a graph using more complicated combinatorial structures than just edges. In this context, the standard matching can be thought of as a packing with edges, i.e. a $\{K_2\}$-packing. The study of packing has a lot in common with our work as it deals with nontrivial extensions of matching, often using totally different methods. For some results on packings, see for example [10, 24, 27, 28, 46, 52, 63], for surveys see [26, 29, 47]. Especially relevant to our work is packing by edges and triangles ($\{K_2, K_3\}$ packing), since choosing a 3d edge in *Simplex Matching* is similar to choosing a triangle for a packing. In [48], Hell and Kirkpatrick gave an elegant algorithm to find the perfect $\{K_2, K_3\}$-packing in unweighted graphs, and [54] classified some types of packings that can be found efficiently. Hell and Kirkpatrick's algorithm can be easily extended to solve the unweighted version of *Simplex Matching*. The weighted case is significantly more complicated, however, and cannot be solved by any simple extension of the unweighted algorithm. Since *Simplex Matching* is a generalization of $\{K_2, K_3\}$ packing, our algorithm gives a simple and intuitive way to find the minimum cost $\{K_2, K_3\}$ perfect packing. The algorithm we provide is relatively simple to understand and implement but difficult to prove correct. In the process of the proof we show some new results about covering cubic graphs with simple combinatorial objects. Another relevant line of research is packing by cycles of length *at least* three [17, 18].

Much of the literature on packing concerns itself with matching polyhedra. Unlike the standard perfect matching, which has a nice characterization as a linear program, many similar results for packing can be extremely complicated. While the weighted perfect matching problem lends itself to a primal-dual algorithm, this is not true for *Simplex Matching*, for which there is no nice linear program characterizing the solutions. [26, 29, 47, 54] give polytope characterizations of other packing problems, although most of these are either for unweighted versions, or for packing problems very different from ours. An exception is Gyula Pap, who produces results similar to ours in [64] (including an efficient algorithm for weighted $\{K_2, K_3\}$ packing), although he uses different techniques and considers the problem from a different perspective.

Finally, Xu et al. [74] discuss the *Terminal Backup* problem in detail and provide an implementation of our *Simplex Matching* algorithm together with some optimizations that are important if the algorithm is to be actually used in practice.

4.5 Our Contribution

Our main contribution consists of providing a polynomial-time algorithm for *Simplex Matching*, which can be used to solve a variety of related problems. The algorithm is very simple conceptually. It starts with a perfect matching (packing) M, and at every step finds an M-alternating 2-factor,[1] such that augmenting M by this 2-factor creates a significantly cheaper perfect matching. It is not surprising that such an algorithm exists, since the minimum cost perfect matching can be obtained from any perfect matching if we just augment it by the correct 2-factor. What is surprising here is that a desirable 2-factor can be found efficiently.

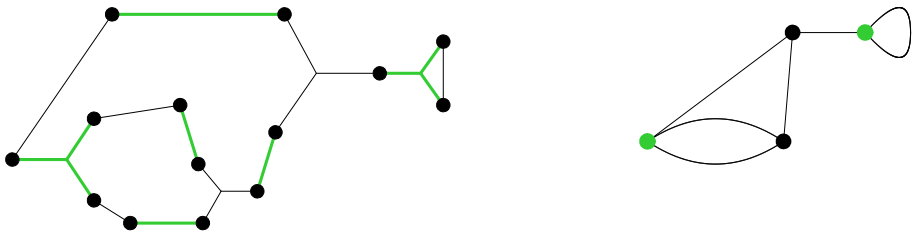


Figure 4.3: (Left) An M-alternating 2-factor. The green edges are in the matching M. 3d edges are drawn as a star with 3 leaves (i.e., the nodes in the middle of these stars are not real nodes). (Right) The dual of this 2-factor (see Section 5.3). The green nodes are in M.

Consider how a similar algorithm would behave if we wanted to find the minimum cost perfect matching without any edges of size 3. Then any 2-factor is simply a collection of cycles, and we could find an alternating cycle that decreases the matching cost sufficiently. In the case of *Simplex Matching*, however, the 2-factors can have very complex structure (see Figure 4.3) and finding a good M-alternating 2-factor may seem difficult.

[1] Recall that a 2-factor is a subgraph with every node in this subgraph having exactly 2 incident edges.

To get around this problem, we show that there is no need to consider arbitrary 2-factors like the one in Figure 4.3, as there always exist good 2-factors with simple structure (containing at most two 3d edges), even in the weighted case. The proof of this is complex and relies on our theorem about covering arbitrary cubic graphs with simple combinatorial objects we call *dual augmentors*. For discussion on the relationship between our results and other covering results, especially cycle covers [49, 50, 69, 75], see Section 5.4.

Chapter 5

Simplex Matching

We start by presenting a simple polynomial time algorithm for the unweighted version of *Simplex Matching*, a problem that is of independent interest as a generalization of minimum cost matching, and also provides intuition for and one of the ingredients of our algorithm for weighted *Simplex Matching*. We then present a polynomial time algorithm for weighted *Simplex Matching*. In the course of the analysis of its correctness we show some interesting new results about covering cubic graphs with simple combinatorial objects we call *dual augmentors*.

5.1 An Algorithm for Unweighted *Simplex Matching*

In the unweighted version of *Simplex Matching* we are given a hypergraph $H(V, E)$ with edges of size 2 and 3 such that for every edge $(u, v, w) \in E$, the edges (u, v), (u, w) and (v, w) are also present. We desire to find a perfect matching if one exists. Hell and Kirkpatrick's algorithm for finding the perfect $\{K_2, K_3\}$ packing in unweighted graphs from [48] could be extended to solve the unweighted version of *Simplex Matching*. However, we would still present our algorithm for solving this problem as it provides useful intuition about our approach to solving weighted *Simplex Matching*.

Suppose we currently have a matching M of H. Define the size of the matching to be the number of nodes it covers. We say that a path is M-alternating if each vertex of the

path is either an endpoint of the path or is adjacent to exactly one edge from M and one edge not from M. If there were no 3d edges and M was not a matching of maximum size, we could always find an M-alternating path with endpoints not covered by M. We could form a larger matching by substituting the edges of the augmenting path that are in M with the ones that are not. We would like to prove that we can always augment M even in the presence of 3d edges, except that the augmentation may not be with respect to a path. We now define three types of M-alternating augmenting structures that we call *unweighted augmentors* (see Figure 5.1):

Type-0: A Type-0 unweighted augmentor is an M-alternating path of 2d edges starting and ending at nodes not covered by M.

Type-1: Let P be an M-alternating path starting with a node x not covered by M and ending at a node v incident to a 3d-edge $(v, u, w) \in M$. A Type-1 unweighted augmentor consists of P, (v, u, w) and (u, w).

Type-2: Let x be a node not covered by M and $(u, v, w) \notin M$. Let P_1 and P_2 be two disjoint M-alternating paths, P_1 connecting x to u and P_2 connecting v to w. A Type-2 unweighted augmentor consists of P_1, P_2 and (u, v, w).

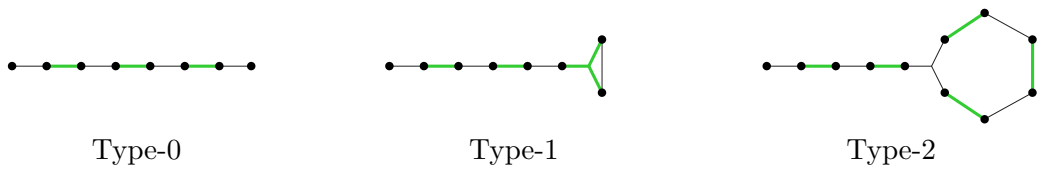


Figure 5.1: *Simplex Matching* Unweighted Augmentors. Edges in green are in M.

Notice that a Type-0 unweighted augmentor is a normal matching augmenting path. We can augment the matching M by replacing the edges of the path that are in M with those not in M to obtain a new matching that covers two extra vertices. Similarly when we augment M by a Type-1 or Type-2 unweighted augmentor, we obtain a new matching

with an extra matched node. This suggests an intuitive algorithm for finding the matching of largest cardinality:

```
Start with the empty matching.
Repeat until done
  Find an unweighted augmentor of Type-0, Type-1 or Type-2.
  Augment the current matching by it.
```

We will prove the correctness of this algorithm in two steps. First we will show that if a matching does not match as many nodes as the matching of largest cardinality then there exists an unweighted augmentor of Type-0, Type-1 or Type-2 by which we can augment the current matching to obtain a matching that covers more vertices. Then we will show that such an unweighted augmentor can be found efficiently.

Lemma 5.1.1 *Let M^* be a maximum matching and let M be a smaller matching. Then there exists an unweighted augmentor that can be used to augment M to a larger matching.*

Proof: Suppose that there is a vertex u matched in M by an edge (u, v) such that u is not matched in M^*. Since M^* matches the most nodes, v is matched in M^* by means of a 2d edge (v, w), otherwise M^* could be augmented to form a larger matching that includes u as well. When thinking of how to augment M possibly using edges in M^* we will ignore the vertices u, v and w. More specifically we will show that there exists an unweighted augmentor in $H - \{u, v, w\}$ that can be used to augment M to a larger matching. We can repeat this elimination of vertices until there is no vertex u matched in M by a 2d edge but not matched in M^*. The portions of M^* and M restricted to this "pruned" hypergraph will be denoted by $\hat{M}^*$ and $\hat{M}$ respectively.

Consider the symmetric difference $\hat{M}^* \oplus \hat{M}$ of the two pruned matchings. Let E^{extra} be the set of 2d edges completing the simplex of each 3d edge in $\hat{M}^*$, i.e. $E^{extra} = \{(u, v) \mid \exists\, w \in V$ such that $(u, v, w) \in \hat{M}^*\}$. Let $S = (\hat{M}^* \oplus \hat{M}) \cup E^{extra}$. Since $\hat{M}$ is a smaller matching

than $\hat{M}^*$ there exists some node v not covered by $\hat{M}$ that is the endpoint of an edge in S. If there exists a $\hat{M}$-alternating path starting at v and ending either at another node not covered by $\hat{M}$ or at a 3d edge in $\hat{M}$, then we have a Type-0 or Type-1 unweighted augmentor and we are done. Therefore we can assume that all $\hat{M}$-alternating paths (in S) starting at v must either end at a 3d edge of $\hat{M}^*$, or eventually encounter an already visited vertex and form a cycle. All paths cannot end at a 3d edge of $\hat{M}^*$, since we can always take an edge in E^{extra} instead of the 3d edge and continue the path in this manner. This means that there must be some $\hat{M}$-alternating path P starting from v that eventually encounters an already visited vertex u (see Figure 5.2). The vertex u is incident to edges

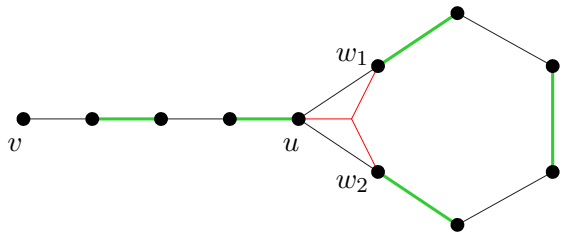


Figure 5.2: An $\hat{M}$-alternating path P that encounters an already visited vertex u. The green edges are in $\hat{M}$, the red edge is in $\hat{M}^*$.

e_1 and e_2 that are in S but not in $\hat{M}$. Since $\hat{M}^*$ is a matching, one of these edges (say e_1) is in E^{extra}. By the definition of E^{extra}, there must be a 3d-edge $(u, w_1, w_2) \in \hat{M}^*$. Since $\hat{M}^*$ is a matching, then $e_2 \in E^{extra}$ as well, and it must be that e_1 and e_2 are in fact (u, w_1) and (u, w_2). Therefore, we can replace e_1 and e_2 with (u, w_1, w_2) in P to form a Type-2 unweighted augmentor. □

It is now sufficient to show that an unweighted augmentor can be found efficiently. We start with the following structure lemma:

Lemma 5.1.2 *If G is an undirected graph (without 3d edges) and a matching M matches all but $2k$ nodes of G then G has a perfect matching if and only if there are k edge-disjoint M-alternating paths joining all $2k$ nodes together.*

Proof: If there are k such edge-disjoint paths, then we can augment M with these paths, converting M into a perfect matching, as desired.

Conversely, suppose G has a perfect matching, and there are at most $l < k$ edge-disjoint M-alternating paths $P_1, P_2, \ldots, P_l$ connecting $2l$ of the unmatched nodes together. Notice that these paths are also vertex-disjoint, since M is a matching and they are M-alternating. Form a new matching M' by augmenting M with $P = P_1 \cup P_2 \cup \ldots \cup P_l$. Since M' is not a perfect matching, there must exist an M'-alternating path P_{l+1} connecting two vertices not matched by M'. If P_{l+1} is disjoint from P, then we are done. Otherwise, consider the symmetric difference $P' = P \oplus P_{l+1}$. Let $d_{P'}(v)$ be the degree of a vertex v in P', i.e., the number of edges in P' incident to v. Note that since the degree of vertices in $P \cup P_{l+1}$ is at most 3 then $d_{P'} \leq 3$ (P_{l+1} is M'-alternating and if it intersects some $P_i \in P$ at a node then it must also intersect it at an edge).

If v is not matched by M but is touched by P or P_{l+1} then $d_{P'}(v) = 1$. If $d_{P'}(v)$ were higher then it would have to be at least 2 in $P \cup P_{l+1}$, which means it must have a path P_i from P starting at it and P_{l+1} going through it. Since P_{l+1} is M'-alternating and the end edges of all paths in P are in M', then it must be that P_{l+1} contains the edge of P_i incident to v. It follows that in this case $d_{P'}(v)$ is no more than 1, a contradiction. If $d_{P'}(v) = 0$, then all the edges in $P \cup P_{l+1}$ incident to v are in $P \cap P_{l+1}$. P has at most one edge incident to v, which implies that v is an endpoint of some $P_i \in P$ as well as of P_{l+1}. But the endpoints of P_{l+1} are not matched by M', while all endpoints of paths in P are matched by M' by construction, a contradiction. Therefore, there are $2l + 2$ vertices not matched by M with degree exactly 1 in P'.

We will show that the degree of all other vertices in P' is 0 or 2 and that for each vertex v such that $d_{P'}(v) = 2$ exactly one edge of P' incident to v is in M.

Let v be a vertex contained in some $P_i \in P$ and let e_1, e_2 be the edges of P_i incident to v. If neither e_1 nor e_2 are in P_{l+1} then clearly $d_{P'}(v) = 2$ and $d_{P' \cap M}(v) = 1$ since P_i is M-alternating. If both are in P_{l+1} then $d_{P'}(v) = 0$. If only one of e_1 and e_2 is in P_{l+1} (say e_1) then there exists an edge e_3 in P_{l+1} but not in P with endpoint v, resulting in

$d_{P'}(v) = 2$. P_i is M-alternating so either e_1 or e_2 is in M. If $e_1 \in M$ then $e_2 \in M'$ and $e_1 \notin M'$ by the construction of M'. Since M' is a matching, $e_3 \notin M'$ which contradicts the fact that P_{l+1} is M'-alternating. Therefore it must be that $e_2 \in M$ and $d_{P' \cap M}(v) = 1$ as desired. Finally, if $v \in P_{l+1}$ and e_1, e_2 are edges in P_{l+1} that are not in P, then $d_{P'}(v) = 2$ and $d_{P' \cap M}(v) = 1$ since P_{l+1} is M'-alternating, and M' equals M everywhere except P.

We now know that P' consists of nodes of degree 1 with incident edges not in the matching M and of nodes of degree 2 with exactly one incident edge in M. Therefore, P' is a collection of disjoint M-alternating paths and cycles joining $2l + 2$ nodes unmatched by M. This proves the lemma, since we now have a method of constructing $l + 1$ desired paths from l, as long as there exists a matching with larger cardinality. □

We finally have all the ingredients to find the largest matching of H in polynomial time.

Theorem 5.1.3 *Given an unweighted graph H with 3d edges as defined above, we can find the largest matching in polynomial time.*

Proof: We start with an empty matching M and keep augmenting it by unweighted augmentors. By Lemma 5.1.1 it is enough to show that we can find an unweighted augmentor in polynomial time if one exists, since one will always exist if M is not maximal. We can find a Type-0 and Type-1 unweighted augmentors using the standard "blossom-contracting" algorithm of Edmonds [31]. The only thing left to show is that if M is not a maximal matching and no Type-0 or Type-1 unweighted augmentors exist we can find a Type-2 unweighted augmentor efficiently.

Fix a 3d-edge $(u_1, u_2, u_3) \notin M$, and a vertex v not matched by M (we can enumerate all such combinations). Now form a new graph $G' = (V', E')$ by removing all 3d edges from H and adding three extra vertices u'_1, u'_2, u'_3 and the extra edges (u'_i, u_i). Let M' be the portion of M that consists only of 2d edges. For every vertex w not matched in M' except for $\{v, u_1, u_2, u_3\}$ add a vertex w' with an edge (w', w) to the new graph, and add the edge (w', w) to M'.

Suppose there is a Type-2 unweighted augmentor connecting v to u_1 and u_1 to u_2. Then

there are disjoint M'-alternating paths in G' connecting v to u'_1 and u'_2 to u'_3. Conversely, if such disjoint M'-alternating paths exist, then we can find a Type-2 unweighted augmentor. Using Lemma 5.1.2 we can find these disjoint paths if such an unweighted augmentor exists (using the algorithm for perfect matching, or using the proof of 5.1.2) and therefore we can find the Type-2 unweighted augmentor itself. □

The number of iterations of matching augmentation is $O(n)$ where n is the number of vertices in the graph. During each iteration we first check for Type-0 and Type-1 unweighted augmentors in $O(n^2)$ time. If none are found we enumerate all pairs of a 3d edge and a vertex (nm combinations where m is the number of edges in the graph) and look for two disjoint M'-alternating paths in time $O(nm)$. Thus the final running time of the algorithm is $O(n^3m^2)$.

5.2 An Algorithm for Weighted *Simplex Matching*

Consider the problem of finding the perfect matching of minimum cost for the standard case where all edges are of size 2. If a perfect matching M is not the minimum-weight one, then there exists an alternating cycle of edges in M and edges not in M that could be used to improve the current matching. We now show a similar condition for *Simplex Matching*.

Let M be a perfect matching of cost $\sum_{e\in M} c(e)$. For any set of edges S, define a potential function $\phi_M(S) = \sum_{e\in M\cap S} c(e) - \sum_{e\in S-M} c(e)$. If we augment M by S by replacing all edges in $M \cap S$ with the edges in $S - M$, the cost of the new set decreases by $\phi_M(S)$. Moreover, if S is an M-alternating 2-factor then this is still a perfect matching.[1] Let M^* be a minimum cost perfect matching. The components of the symmetric difference $M \oplus M^*$ are M-alternating 2-factors that augment M to M^*. Therefore, there always exists an M-alternating 2-factor S with $\phi_M(S) \geq 0$. This suggests the following very intuitive algorithm for finding the minimum cost weighted *Simplex Matching*:

[1] An M-alternating 2-factor is a set S such that every node in S has exactly two edges incident to it, exactly one of which is in M.

```
Start with any perfect matching.
Repeat until done
  Find an alternating 2-factor with positive potential.
  Augment the current matching by it.
```

If we could find the M-alternating 2-factor with maximum potential ϕ_M, we could simply augment by it and get the minimum cost perfect matching. Instead, our algorithm will proceed by finding an M-alternating 2-factor S with high $\phi_M(S)$ at each step, and augmenting by it. Finding a 2-factor S with a high potential $\phi_M(S)$ seems difficult, since 2-factors for *Simplex Matching* can have a complex structure, as in Figure 4.3. We will show, however, that there is no need to consider arbitrary 2-factors like that, because there always exists a good 2-factor with simple structure: it should contain at most two 3d edges. We call such 2-factors *augmentors*. More specifically, augmentors can be of the following types (see Figure 5.3):

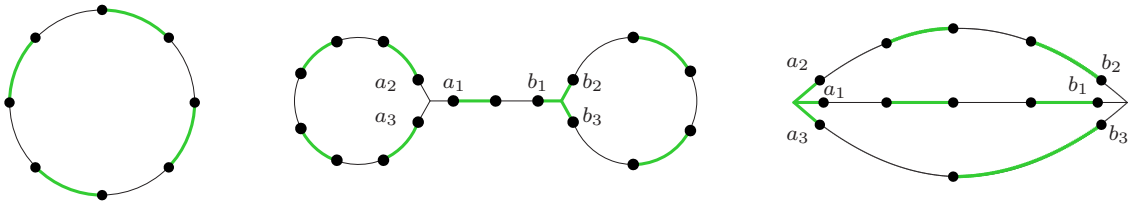
Type-0: A Type-0 augmentor is an M-alternating cycle of 2d edges. This is the same as a 2d matching augmenting cycle.

Type-1: A Type-1 augmentor consists of two 3d edges (a_1, a_2, a_3) and (b_1, b_2, b_3) together with M-alternating paths of 2d edges connecting a_1 to b_1, a_2 to a_3 and b_2 to b_3. These paths must be disjoint and the entire augmentor must be M-alternating (so the 3d edges may or may not be in M).

Type-2: Same as Type-1, but the paths connect a_1 to b_1, a_2 to b_2 and a_3 to b_3.

The majority of our work is devoted to proving that an augmentor with high potential always exists. The following lemma allows us to claim that if this is true, then we can improve the current perfect matching in polynomial time.

Lemma 5.2.1 *Let A be an augmentor of maximum potential. We can find an M-alternating 2-factor S with $\phi_M(S) \geq \phi_M(A)$ in polynomial time (that may or may not be A itself).*



Type-0 Type-1 Type-2

Figure 5.3: *Simplex Matching* Augmentors

Proof: Suppose A is of Type-0. Delete all 3d edges from H as well as all nodes that are matched in M using 3d edges, forming a graph H' with no 3d edges. Find a minimum cost matching M^* of H'. $S = M^* \oplus M \mid_{H'}$ gives us an M-alternating 2-factor to augment M by. Since A is of Type-0 it cannot include any nodes incident to 3d edges of M, thus A is contained in H'. Augmenting by S results in the best possible matching in H', therefore $\phi_M(S) \geq \phi_M(A)$.

Now suppose A is of Type-1 or Type-2. Fix the two 3d edges e_1 and e_2 that it contains (there are only $|E|^2$ possibilities). We will find the best M-alternating 2-factor with only e_1 and e_2 as the 3d edges. To do this we form a new graph H' as above, except we leave the nodes incident to $e_1 = (u_1, u_2, u_3)$ and $e_2 = (v_1, v_2, v_3)$ in H'. If $e_1 \notin M$, form three new nodes u'_1, u'_2, u'_3 with edges (u_i, u'_i) of cost 0 and do likewise for e_2. There are only six unmatched nodes in H': either u_1, u_2, u_3 if $e_1 \in M$, or u'_1, u'_2, u'_3 if $e_1 \notin M$, and similarly for e_2. Now find a minimum cost matching M^* of H'. $M^* \oplus M \mid_{H'}$ is an M-alternating 2-factor except for the nodes that were unmatched in H' using M, which have degree 1. Add e_1 and e_2 to $M^* \oplus M \mid_{H'}$ (if $e_1 \notin M$, then adding it replaces the edges (u_i, u'_i)), forming an M-alternating 2-factor S. As before, A is a possible way to augment M in H' and since S is the best such way we have $\phi_M(S) \geq \phi_M(A)$.

We now choose the best one of the resulting $|E|^2 + 1$ M-alternating 2-factors. □

There are many ways to make the above algorithm run faster in practice. Notice that we do not need to consider pairs of 3d edges (e_1, e_2) if $e_1 \notin M$ is adjacent to a 3d edge of M that is not e_2. For more improvements, see Section 5.5 and [74].

Using Lemma 5.2.1, we can restate our algorithm for finding the minimum cost weighted *Simplex Matching* as shown below. The initial perfect matching can be found by the algorithm for unweighted *Simplex Matching* described in the previous section.

```
Start with any perfect matching.
Repeat until done
  Find an alternating 2-factor using Lemma 5.2.1.
  Augment the current matching by the above 2-factor.
```

5.3 Dual Augmentors

We establish the running time of the above algorithm in Section 5.5, and now focus on its correctness and termination. To prove this, we need to show that for any perfect matching M that is not of minimum cost there exists an augmentor A with $\phi_M(A) > 0$. We will accomplish this by showing that every M-alternating 2-factor of positive potential contains an augmentor of positive potential.

For any S we form a dual graph $\overline{S}$ that is easier to deal with than S (see Figures 4.3 and 5.4). First, we contract all 2d edges (u, v) of S such that u and v are not part of the same 3d edge. Then we replace each 3d edge e with a node v_e. We then form an edge between the new nodes if the 3d edges that produced them were adjacent. Note that this may result in parallel edges as well as self-loops (self-loops occur if both (u, v, w) and (v, w) were in S). The resulting graph $\overline{S}$ is a cubic (3-regular) graph. We will say that a node $v \in \overline{S}$ is in M if its corresponding 3d edge of S is in M.

Let S_{extra} be the multiset of 2d edges (u, v) such that some 3d edge (u, v, w) is in S, but $(u, v) \notin S$, as in Figure 5.4. Let $S' = S \cup S_{extra}$. We will associate a unique object from $\overline{S}$ that we call a *dual augmentor* with each augmentor A in S'. The dual augmentor is simply the subgraph of $\overline{S}$ corresponding to the dual of A, as shown in Figure 5.4. We explore exactly what this means below, but the reader can look at Lemma 5.3.1 or Figure 5.5 for a compact definition of a dual augmentor. Dual augmentors are simply one of the

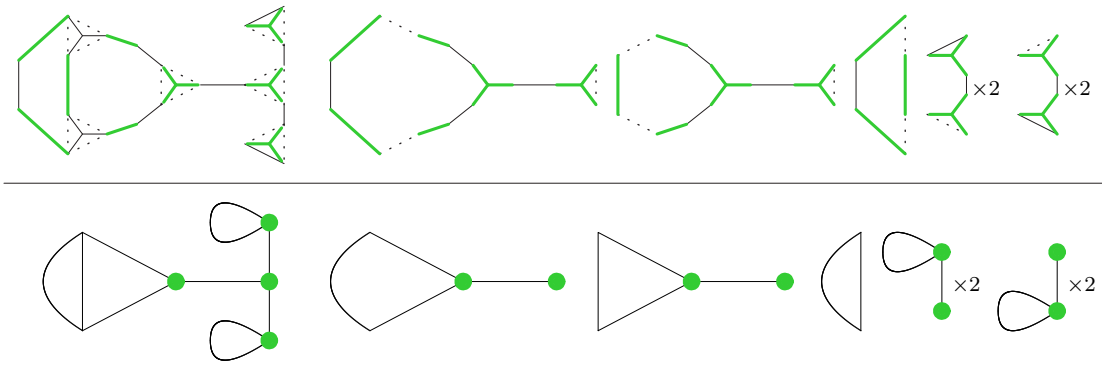


Figure 5.4: (Top) An M-alternating 2-factor S with edges of S_{extra} shown as dashed lines. To the right of it are some augmentors. (Bottom) The dual cubic graph $\overline{S}$, and the corresponding dual augmentors. The nodes of M are in green.

structures in Figure 5.5, with nodes of M having degree 1 or 3 (not 2).

Type-0: A Type-0 augmentor A of S' is an M-alternating cycle of 2d edges, some of which are in S_{extra}, like the third augmentor shown in Figure 5.4. After contracting all 2d edges of S, A becomes a cycle of S_{extra} edges. In the dual graph $\overline{S}$, A corresponds to a cycle of nodes v_e where each e is the 3d edge that produced one of the above S_{extra} edges. We refer to these cycles as dual augmentors of Type-0. As A is M-alternating, it cannot be that any of the above v_e is in M, since then $e \in M$, which would mean that the endpoints of some S_{extra} edge in A are contained in two edges of M.

Type-1: A Type-1 augmentor A will produce a dual augmentor of one of three kinds, as shown in Figure 5.5. The M-alternating paths connecting a_1 to b_1, a_2 to a_3, and b_2 to b_3 behave exactly as the cycle augmentor in the previous case, namely they correspond to paths that do not contain vertices of M. This gives us a dual augmentor of Type-1c in Figure 5.5 that is a path together with the two cycles attached to it, with only nodes $v_{(a_1,a_2,a_3)}$ and $v_{(b_1,b_2,b_3)}$ possibly being in M.

Notice that if both (a_1, a_2, a_3) and (a_2, a_3) are in A, then the "cycle" incident to $v_{(a_1,a_2,a_3)}$ in the dual augmentor is just a self-loop. Consider the special case, however,

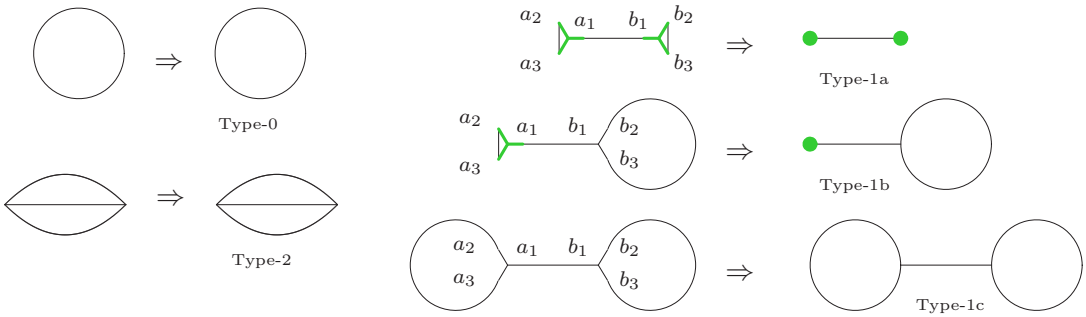


Figure 5.5: Transforming augmentors into dual augmentors.

when $(a_1, a_2, a_3) \in M$ and $(a_2, a_3) \in S_{extra}$, as in many augmentors of Figure 5.4. We cannot form a self-loop in the dual augmentor, because we only formed self-loops in $\overline{S}$ for edges of S, not S_{extra}. Because of this, we simply have no loop at all, and we associate to A a dual augmentor of Type-1a or Type-1b (depending if this special case occurs at both (a_1, a_2, a_3) and (b_1, b_2, b_3) or just one of them). Notice that only nodes that are in M can have degree 1 in a dual augmentor.

Type-2: By similar reasoning, we obtain a Type-2 dual augmentor shown in Figure 5.5, with only the degree 3 nodes possibly being in M.

It is easy to see that the following lemma holds.

Lemma 5.3.1 *There is a one-to-one correspondence between augmentors in S' and dual augmentors in the cubic graph $\overline{S}$, and dual augmentors satisfy the following conditions (in fact, these conditions are an alternate definition of dual augmentors):*

1. *Degree 2 everywhere except at most 2 nodes.*
2. *All degree 1 nodes are in M.*
3. *All degree 2 nodes are not in M.*

In other words, dual augmentors are the structures in Figure 5.5, with the only nodes in M being the ones of degree 1 or 3. The reason for considering the dual graph $\overline{S}$ instead of

S is that we now have a cubic (i.e., 3-regular) graph and as the lemma below will show, our goal now will be to cover this cubic graph with dual augmentors. While the same results can be proven directly for S' instead of $\overline{S}$, their statements become a lot more complicated.

We now proceed to argue that there always exists an augmentor with high potential, for which we need the concept of *augmentor sum*. Let $\overline{\mathcal{A}}$ denote the set of all possible dual augmentors contained in $\overline{S}$. Then

Definition 5.3.2 ***(valid augmentor sum)*** *A function $\alpha : \overline{\mathcal{A}} \to \mathbb{N}$ is a* valid augmentor sum *of $\overline{S}$ if and only if there exists $x > 0$ such that for all edges e of $\overline{S}$, we have $\sum_{A \in \overline{\mathcal{A}}, A \ni e} \alpha(A) = x$.*

In other words, a valid augmentor sum is a cover of $\overline{S}$ with dual augmentors so that every edge is contained in exactly the same number of elements (which we call the *cover number*). Given that there is a one-to-one correspondence between augmentors in S' and dual augmentors in $\overline{S}$ we can also view α as a weight assignment on the augmentors in S'. Figure 5.4 illustrates a set of dual augmentors that form a valid augmentor sum by covering every edge of $\overline{S}$ twice, as well as the augmentors of S' they correspond to. The lemma below shows that if $\phi_M(S) > 0$, then the same must be true for at least one of the augmentors in that list. The idea behind it is that if all augmentors corresponding to the dual augmentors in α "add up" to S and S is improving, then so is some augmentor in the sum.

Lemma 5.3.3 *Given a perfect matching M and an M-alternating 2-factor S such that $\phi_M(S) > 0$, there exists an augmentor A with $\phi_M(A) \geq \frac{\phi_M(S)}{|S|}$ if there exists a valid augmentor sum α of $\overline{S}$.*

Proof: If we had a cover of S by augmentors, such that every edge of S is contained in the same number of augmentors, then we would immediately know that some augmentor must have positive potential. This follows easily since the total potential of the augmentors must equal a multiple of $\phi_M(S)$. Unfortunately, we have such a cover of $\overline{S}$ and not S. As shown in Figure 5.4, dual augmentors of $\overline{S}$ can correspond to augmentors that include edges in S_{extra} and not S. In fact, if we look at Figure 5.4 we can see that there are some 3d edges

of S that are not contained in any augmentors from the list, even if this list forms a valid augmentor sum of $\overline{S}$ with a cover number of 2. Notice, however, that the edges of S_{extra} corresponding to these 3d edges *are* included in the list of augmentors, which we will be able to relate to the cost of the 3d edges using the *Simplex Condition*.

Let x be the cover number of $\alpha : \overline{\mathcal{A}} \to \mathbb{N}$ and let $\mathcal{A}$ be the set of augmentors in S'. Since there is a one-to-one correspondence between $\overline{\mathcal{A}}$ and $\mathcal{A}$ we will consider α as an integer weight assignment to augmentors in S'. First we will compute how many times α covers an edge in $S \subseteq S'$.

1. If $e = (u, v) \in S$ then $\sum_{A \in \mathcal{A}, A \ni e} \alpha(A) = x$.

   Proof: A 2d edge in S' corresponds to some edge in $\overline{S}$, which is covered exactly x times.

2. If $e = (u, v, w) \in S$ such that some $(u, v) \in S$ then $\sum_{A \in \mathcal{A}, A \ni e} \alpha(A) = x$.

   Proof: The condition implies that the node v_e corresponding to e in $\overline{S}$ has a self-loop. By construction, all dual augmentors containing v_e must also contain the self-loop, which is covered x times, so e is covered exactly x times.

3. If $e = (u, v, w) \in M \cap S$ such that $e_1 = (u, v), e_2 = (v, w), e_3 = (u, w) \notin S$ then $\forall i$, $\sum_{A \in \mathcal{A}, A \ni e} \alpha(A) - 2 \sum_{A \in \mathcal{A}, A \ni e_i} \alpha(A) = x$.

   Proof: Consider the middle rightmost 3d edge of Figure 5.4. It appears in several augmentors, with different corresponding edges from S_{extra}. What the above statement implies is that each of these edges in S_{extra} are covered the same number of times.

   Let v_e be the node in $\overline{S}$ corresponding to e. Each edge incident to v_e appears in dual augmentors exactly x times and each dual augmentor either contains all three edges or exactly one of them, since the degree of a node in M is 1 or 3. Let a be the number of dual augmentors in α containing all 3 edges incident to v_e. Then all remaining dual augmentors that contain v_e must be organized as a set of size $x - a$ of triples such that each dual augmentor in the triple contains a different edge incident

to v_e. Every dual augmentor like this corresponds to an augmentor of S' that contains the edge e and also contains exactly one of e_1, e_2 or e_3 in S_{extra}. The a augmentors and the $x - a$ triples of augmentors all contain e in S', and so edge e is contained in $a + 3(x - a)$ augmentors, and each e_i is contained in $x - a$ augmentors, producing the desired result.

4. If $e = (u, v, w) \in S - M$ such that $e_1 = (u, v), e_2 = (v, w), e_3 = (u, w) \notin S$ then $\forall i$, $\sum_{A \in \mathcal{A}, A \ni e} \alpha(A) + 2 \sum_{A \in \mathcal{A}, A \ni e_i} \alpha(A) = x$.

   Proof: Let v_e be the node in $\overline{S}$ corresponding to e. Each edge incident to v_e appears in dual augmentors exactly x times and each dual augmentor either contains all three edges or exactly two of them, since the degree of v_e in a dual augmentor can only be 2 or 3. Let a be the number of dual augmentors in α containing all 3 edges incident to v_e. These correspond exactly to the augmentors containing e in S'. The rest of the dual augmentors that cover edges adjacent to v_e must be organized as a set of size $(x - a)/2$ of triples such that each dual augmentor in the triple contains two different edges incident to v_e, so that the entire triple covers each edge twice. A corresponding triple of augmentors in S' contains each of e_1, e_2 and e_3 once. Therefore, edge e is contained in a augmentors, and each edge e_i is contained in $(x - a)/2$, producing the result.

Using these covering results we can now bound $x\ \phi_M(S) = x \sum_{e \in S} \phi_M(e)$. If $e = (u, v)$, or $e = (u, v, w) \in S$ with some $(u, v) \in S$, we have $x\ \phi_M(e) = \sum_{A \in \mathcal{A}, A \ni e} \alpha(A) \phi_M(e)$. If $e = (u, v, w) \in M \cap S$ such that $e_1 = (u, v), e_2 = (v, w), e_3 = (u, w) \notin S$, then $\phi_M(e) = c(e)$

and $\phi_M(e_i) = -c(e_i)$, so

$$\begin{aligned} x\ \phi_M(e) &= \sum_{A\in\mathcal{A},A\ni e} \alpha(A)\phi_M(e) - 2\sum_{A\in\mathcal{A},A\ni e_i} \alpha(A)\phi_M(e) \leq \\ &\sum_{A\in\mathcal{A},A\ni e} \alpha(A)\phi_M(e) - \sum_{A\in\mathcal{A},A\ni e_i} \alpha(A)(c(e_1)+c(e_2)+c(e_3)) = \\ &\sum_{A\in\mathcal{A},A\ni e} \alpha(A)\phi_M(e) + \sum_{A\in\mathcal{A},A\ni e_i} \alpha(A)(\phi_M(e_1)+\phi_M(e_2)+\phi_M(e_3)) = \\ &\sum_{A\in\mathcal{A},A\ni e} \alpha(A)\phi_M(e) + \sum_{A\in\mathcal{A},A\ni e_1} \alpha(A)\phi_M(e_1) + \\ &\sum_{A\in\mathcal{A},A\ni e_2} \alpha(A)\phi_M(e_2) + \sum_{A\in\mathcal{A},A\ni e_3} \alpha(A)\phi_M(e_3) \end{aligned}$$

The inequality above holds because of the *Simplex Condition* on e and the last equality because $\sum_{A\in\mathcal{A},A\ni e_i} \alpha(A)$ is the same for all $i = 1, 2, 3$.

Similarly, if $e = (u, v, w) \in S - M$ such that $e_1 = (u, v), e_2 = (v, w), e_3 = (u, w) \notin S$, then $\phi_M(e) = -c(e)$ and $\phi_M(e_i) = -c(e_i)$, so

$$\begin{aligned} x\ \phi_M(e) &= \sum_{A\in\mathcal{A},A\ni e} \alpha(A)\phi_M(e) + 2\sum_{A\in\mathcal{A},A\ni e_i} \alpha(A)\phi_M(e) \leq \\ &\sum_{A\in\mathcal{A},A\ni e} \alpha(A)\phi_M(e) - \sum_{A\in\mathcal{A},A\ni e_i} \alpha(A)(c(e_1)+c(e_2)+c(e_3)) = \\ &\sum_{A\in\mathcal{A},A\ni e} \alpha(A)\phi_M(e) + \sum_{A\in\mathcal{A},A\ni e_1} \alpha(A)\phi_M(e_1) + \\ &\sum_{A\in\mathcal{A},A\ni e_2} \alpha(A)\phi_M(e_2) + \sum_{A\in\mathcal{A},A\ni e_3} \alpha(A)\phi_M(e_3) \end{aligned}$$

Therefore we have that

$$x\ \phi_M(S) \leq \sum_{A\in\mathcal{A}}\sum_{e\in A} \alpha(A)\phi_M(e) = \sum_{A\in\mathcal{A}} \alpha(A)\phi_M(A)$$

If for all $A \in \mathcal{A}$, $\phi_M(A) < \frac{\phi_M(S)}{|S|}$, then the above inequality shows that $x\ \phi_M(S) < \frac{\phi_M(S)}{|S|}\sum_{A\in\mathcal{A}} \alpha(A)$. Now consider the dual augmentors corresponding to the augmentors A. Every edge in $\overline{S}$ is covered exactly x times, hence $\sum_{A\in\mathcal{A}} \alpha(A) \leq x|E(\overline{S})|$. By the definition of $\overline{S}$ we know that $|E(\overline{S})| \leq |S|$. Overall, this implies that $x\ \phi_M(S) < x\ \phi_M(S)$, giving us a contradiction, as desired. □

Given a perfect matching M that is not minimum cost, we know there exists an M-alternating 2-factor S with $\phi_M(S) > 0$, since every component of the symmetric difference

of M with the minimum cost perfect matching is such a 2-factor. By Lemma 5.3.3, it is enough to show that $\overline{S}$ has a valid augmentor sum to prove that our algorithm finds the minimum cost perfect matching. The following theorem completes the correctness proof.

Theorem 5.3.4 *Any cubic multigraph $\overline{S}$ (possibly with self-loops) has a valid augmentor sum α with respect to any set of nodes M.*

5.4 Valid Augmentor Sums

In this section we can set aside our algorithm and *Simplex Matching*, and concentrate on proving Theorem 5.3.4. We assume that we are given an arbitrary cubic multigraph $\overline{S}$ that may contain self-loops, and some set M of nodes in $\overline{S}$. We show that there always exists a valid augmentor sum of $\overline{S}$ with respect to M.

To understand when such valid sums may exist, consider the special case of $M = \emptyset$. In [69], Seymour states that we can cover any cubic 2-connected graph with cycles so that every edge is in the same number of cycles. Since $M = \emptyset$, all cycles are dual augmentors (they satisfy the conditions of Lemma 5.3.1), and so we always have a valid augmentor sum. There has been much work in finding cycle covers with small cover numbers [49, 75], and it is unknown if there always exists a cycle cover of a cubic 2-connected graph with cover number 2 (this is the Cycle Double Cover Conjecture). Since we are only proving an existence result, however, for our purposes the cover number does not need to be small.

The fact that M may not be empty complicates matters. For example, while forming a cycle cover of a planar graph is easy, consider forming an augmentor sum of Figure 5.10, or of K_4 with $|M| = 1$. Lemma 5.3.1 puts degree constraints on nodes of M, which makes augmentor sums much more difficult to deal with than cycle covers.

We start the proof of Theorem 5.3.4 with an easy lemma about augmentor sums.

Lemma 5.4.1 *Let $\mathcal{B}$ be a collection of edge subsets of $\overline{S}$, and $\beta : \mathcal{B} \to \mathbb{N}$ and x_β be such that for all edges e in $\overline{S}$, $\sum_{B \in \mathcal{B}, B \ni e} \beta(B) = x_\beta$. If every $B \in \mathcal{B}$ has a valid augmentor sum then $\overline{S}$ has a valid augmentor sum.*

Proof: Let α_B be the augmentor sum for $B \in \mathcal{B}$, and let x_B be the cover number of α_B. Let x be the least common multiple of all x_B and $\alpha'_B = \frac{x}{x_B}\alpha_B$, so α'_B is a valid augmentor sum of B with cover number x. α'_B is defined only for dual augmentors in B but we can extend it to the set of all dual augmentors $\overline{\mathcal{A}}$ in $\overline{S}$ by setting $\alpha'_B(A) = 0$ for any dual augmentor A of $\overline{S}$ not contained in B.

Define $\alpha(A) = \sum_{B\in\mathcal{B}} \beta(B)\alpha'_B(A)$. Then for all e in $\overline{S}$

$$\begin{aligned}\sum_{A\in\mathcal{A},A\ni e} \alpha(A) &= \sum_{A\in\mathcal{A},A\ni e}\sum_{B\in\mathcal{B}} \beta(B)\alpha'_B(A) = \\ &= \sum_{B\in\mathcal{B}} \beta(B) \sum_{A\in\mathcal{A},A\ni e} \alpha'_B(A) = \\ &= \sum_{B\in\mathcal{B},B\ni e} \beta(B)\ x = x_\beta x\end{aligned}$$

Since every edge of $\overline{S}$ appears in exactly $x_\beta x$ dual augmentors, α is a valid augmentor sum of $\overline{S}$. □

We will also make use of the following cycle cover theorem, due to Seymour ([69]). The lemma gives a sufficient condition for the existence of a valid cycle sum, which is defined in the same way as a valid augmentor sum.

Theorem 5.4.2 ([69]) *We are given a graph G with capacities $cap(e)$. Let $\mathcal{C}(G)$ be the collection of cycles in G and a* valid circuit sum *of G be a function $\beta : \mathcal{C}(G) \to \mathbb{Q}^+$ such that for every edge e of G, $\sum_{C\in\mathcal{C}(G),C\ni e} \beta(C) = cap(e)$. Then, a valid circuit sum exists if for every cut K, we have that for all $e \in K$, $cap(e) \le \sum_{e'\in K-e} cap(e')$.*

We now proceed to construct a valid augmentor sum of $\overline{S}$. We will prove Theorem 5.3.4 with a series of lemmata that show the existence of a valid augmentor sum of $\overline{S}$ in a different way depending on the structure of $\overline{S}$.

Lemma 5.4.3 *If $\overline{S}$ is 3-connected and $|M| \neq 1$, then there exists a valid augmentor sum of $\overline{S}$.*

Proof: Construct a new graph G by adding an extra node s to $\overline{S}$, together with an edge (s, v) for all $v \in M$ (see Figure 5.6). Associate a capacity $cap(e)$ to all edges e of G. If e is

one of the new edges (s, v), set $cap(e) = 3$, otherwise set $cap(e) = 1$. By Theorem 5.4.2, we know that there exists a valid circuit sum of G iff for all cuts K of G and all edges $e \in K$, $cap(e) \leq \sum_{e' \in K-e} cap(e')$. We now show that this holds true for G.

Any cut K in G contains at least three edges since G is 3-connected. If $e \in K$ is such that $cap(e) = 1$ then the above inequality is trivially satisfied (this also finishes the case of $|M| = 0$). Otherwise, if $e = (s, v)$ and K is the cut $(s, G - s)$ then there is another edge of capacity 3 in K since $|M| \geq 2$. Finally, if $e = (s, v)$ and K is not $(s, G - s)$, then $K - e$ is also a cut in $\overline{S}$ and it contains at least three edges of capacity 1 since $\overline{S}$ is 3-connected.

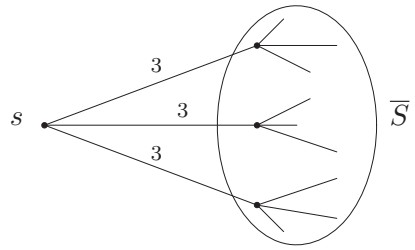


Figure 5.6: Capacitated Graph G in Lemma 5.4.3

We will partition each cycle C with $\beta(C) > 0$ into dual augmentors as follows. Every cycle that does not contain any nodes in M is contained in $\overline{S}$ and is trivially a Type-0 dual augmentor. No cycle passes through two edges with capacity 1 incident to a node $v \in M$, since all other cycles passing through v would not be able to fill (s, v) to its capacity. Therefore, every cycle entering a node in M must proceed to s. Removing all (s, v) edges from these cycles gives us a collection of Type-1a dual augmentors in $\overline{S}$. The valid circuit sum β can be viewed as a fractional weight assignment on those Type-1a and Type-0 dual augmentors such that every edge in $\overline{S}$ is covered exactly once. We can now multiply β by a large enough constant to form a valid augmentor sum. □

Section 5.4.1 addresses the special (and tricky) case when only a single node of $\overline{S}$ is in M. Hence, from this point on we can assume that $\overline{S}$ is not 3-connected. The following two lemmata provide us with augmentor sums for the cases when $\overline{S}$ is 1-connected or 2-connected.

Lemma 5.4.4 *Assume that all cubic multigraphs smaller than $\overline{S}$ have valid augmentor sums and that $\overline{S}$ contains a bridge*[2] *. Then $\overline{S}$ has a valid augmentor sum.*

Proof: Let $e = (u, v)$ be a bridge in $\overline{S}$. Let the other two edges incident to u be $e_1(u)$ and $e_2(u)$, and the other two edges incident to v be $e_1(v)$ and $e_2(v)$. Form two smaller cubic multigraphs S_1 and S_2 by removing e and contracting $e_2(u)$ and $e_2(v)$, as in Figure 5.7. During this contraction, we delete nodes u and v. Notice that in the case where $e_1(u) = e_2(u)$ (u has a self-loop), this process just forms a loop without any nodes, which is a dual augmentor of Type-0 and corresponds to a cycle with no 3d edges in S.

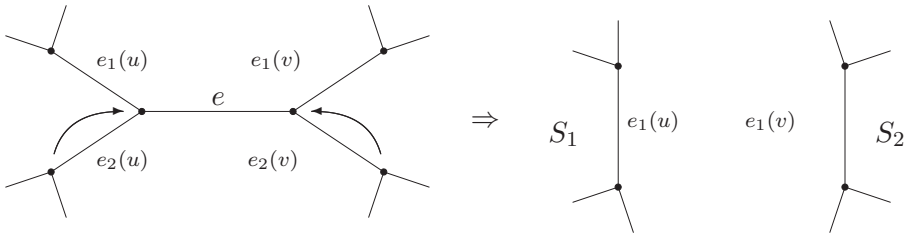


Figure 5.7: Breaking $\overline{S}$ with a bridge into two smaller cubic multigraphs

By the assumption made in the statement of the lemma, there exist valid augmentor sums α_1 and α_2 of S_1 and S_2, with corresponding cover values x_1 and x_2. If x is the least common multiple of x_1 and x_2, then $\frac{x}{x_1}\alpha_1$ and $\frac{x}{x_2}\alpha_1$ are valid augmentor sums of S_1 and S_2 with cover value x. This gives us a multiset of size x of dual augmentors in S_1 that contain $e_1(u)$ and another multiset of size x of dual augmentors in S_2 that contain $e_1(v)$. Pair up the dual augmentors of the S_1 multiset with the dual augmentors of the S_2 multiset and let (A_1, A_2) be one such pair. Consider the multigraph C resulting from adding e to A_1 and A_2, again forming nodes u and v. If C has a valid augmentor sum regardless of what the types of A_1 and A_2 are, then using Lemma 5.4.1 we can deduce that the entire graph $\overline{S}$ has a valid augmentor sum.

Rather than providing a valid augmentor sum individually for all possible graphs C re-

[2]Recall that a bridge is an edge whose removal increases the number of connected components of the graph.

sulting from the pairing of the different types of dual augmentors, we give three simple rules that reduce most of such possible graphs to trivial cases. Each of these rules decomposes C into smaller graphs, such that if all these smaller graphs have valid augmentor sums, then by Lemma 5.4.1 C also has a valid augmentor sum. We give a formal definition of these rules below and Figure 5.8 illustrates them.

Rule 1: Suppose C contains a vertex v of degree 2, the removal of which disconnects C into components C_1 and C_2. If $C - C_1$ and $C - C_2$ have valid augmentor sums, then by Lemma 5.4.1 C also has a valid augmentor sum, as the edge sets of $C - C_1$ and $C - C_2$ cover every edge of C exactly once. We can also apply a similar rule if v is of degree 3. This rule is made for the case where $v \in M$, since v has degree 1 in resulting graphs.

Rule 2: Suppose C contains a vertex v of degree 3, the removal of which disconnects C into components C_1, C_2 and C_3. As above, if $C - C_1$, $C - C_2$ and $C - C_3$ have valid augmentor sums then so does C, as these sets cover every edge of C twice. This rule should be used if $v \notin M$ since v has degree 2 in resulting graphs.

Rule 3: Suppose C contains two vertices u and v of degree 3, the removal of which partitions C into components C_1, C_2, P_1 and P_2 as in Figure 5.8, where P_1 and P_2 are paths. As above, if $C - P_1$, $C - P_2$ and $C - C_1 - C_2$ have valid augmentor sums then so does C, since those graphs cover every edge of C twice. Similarly, if $C - P_1 - C_2$, $C - P_2 - C_2$, $C - C_1 - P_1 - P_2$ and $C - C_1$ have valid augmentor sums then so does C.

We would apply the first variant of Rule 3 when either $u, v \in M$ or $u, v \notin M$, since it is then that the cycle $C - C_1 - C_2$ has a valid augmentor sum (if $u, v \in M$ then it is a sum of two dual augmentors). We would apply the second variant when $u \notin M$ and $v \in M$, since then v always has degree 1 or 3, as needed by Lemma 5.3.1.

It is easy to check that no matter what two structures A_1 and A_2 from Figure 5.5 are stitched together by a bridge to form $C = A_1 \cup A_2 \cup e$, we can always decompose them into dual augmentors by repeated application of the above three rules. The only exception occurs in the example below.

Let A_1 be a dual augmentor of Type-1c, A_2 be a dual augmentor of Type-2 and $u \notin M$

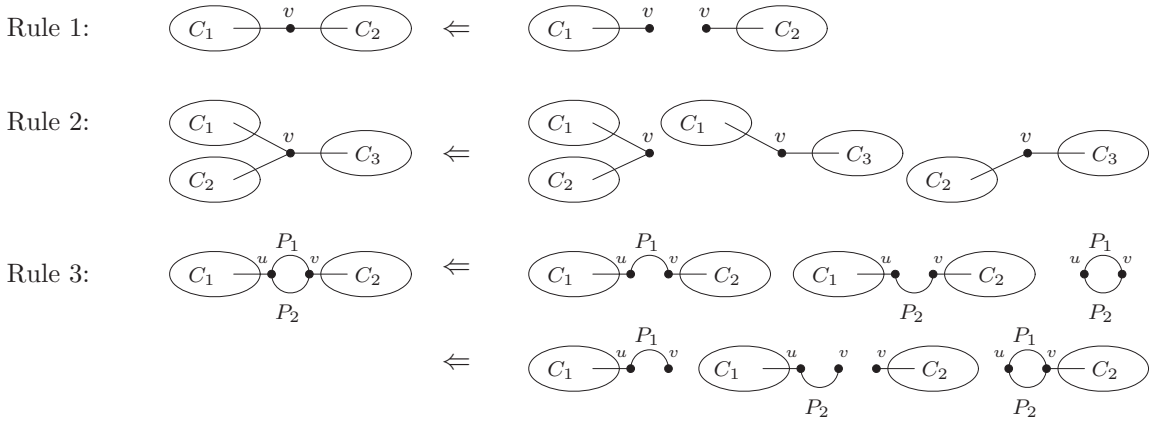


Figure 5.8: Rules for decomposing graphs into easier cases to prove the existence of augmentor sums

lies on one of the cycles of A_1. Moreover, suppose that the node of degree 3 on the same cycle of A_1 is in M. Then we would apply the second version of Rule 3, as shown in Figure 5.9.

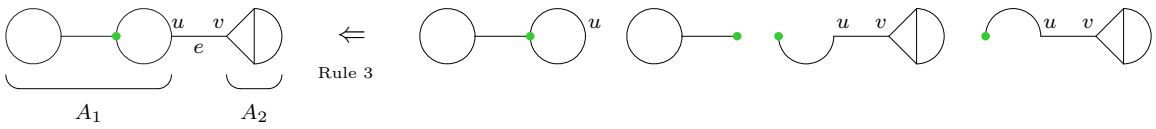


Figure 5.9: Sample reduction for Lemma 5.4.4

Graphs like the last one in Figure 5.9 and ones of similar structure are the only ones that can be formed by this process of repeatedly decomposing C, such that they are neither dual augmentors nor have a further decomposition that is possible using one of the three rules above. For these types of graphs we show a valid augmentor sum directly in Figure 5.10. This completes the proof of the lemma. □

The only case left now is if $\overline{S}$ is bridgeless, but not 3-connected, addressed below.

Lemma 5.4.5 *Assume that all cubic multigraphs smaller than $\overline{S}$ have a valid augmentor sum and that $\overline{S}$ contains two edges e_1 and e_2, the removal of which disconnects it. Then $\overline{S}$ has a valid augmentor sum.*

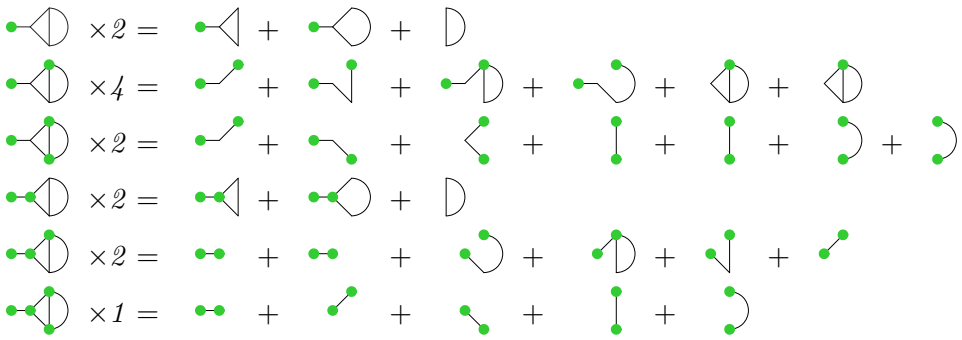


Figure 5.10: Proof that the last remaining case of Lemma 5.4.4 has a valid augmentor sum. Small circles indicate nodes of M, so this shows all possible cases.

Proof: We can assume that $\overline{S}$ is bridgeless. Let $e_1 = (u_1, v_1)$ and $e_2 = (u_2, v_2)$, so that u_1, u_2 are in the same component of $\overline{S} - e_1 - e_2$. We proceed as in the proof of Lemma 5.4.4 to form two smaller cubic multigraphs S_1 and S_2 by removing e_1 and e_2, and forming two new edges (u_1, u_2) and (v_1, v_2), as in Figure 5.11. Notice that the four nodes u_1, u_2, v_1 and v_2 must be distinct, since if $u_1 = u_2$, then the third edge incident to u_1 would be a bridge.

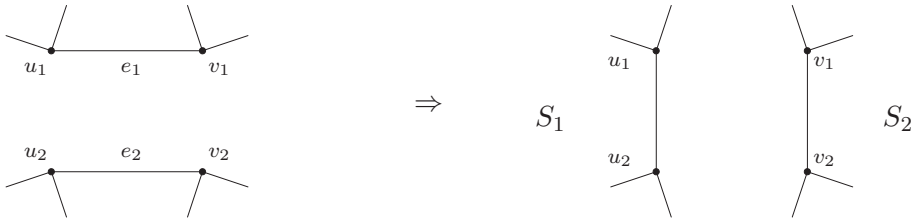


Figure 5.11: Breaking a 2-connected bridgeless $\overline{S}$ into two smaller cubic multigraphs

Similarly to Lemma 5.4.4, there exist valid augmentor sums for S_1 and S_2 with cover value x. This means that there exists a multiset of size x of dual augmentors in S_1 that contain (u_1, u_2) and another multiset of size x of dual augmentors in S_2 that contain (v_1, v_2). Pair up the dual augmentors of the S_1 multiset with the dual augmentors in the S_2 multiset and let (A_1, A_2) be one such pair. Consider the cubic multigraph C resulting from removing

$(u_1, u_2), (v_1, v_2)$ and adding e_1 and e_2 to A_1 and A_2. We will now show that C has a valid augmentor sum, regardless of the types of A_1 and A_2, which finishes the proof using Lemma 5.4.1.

Figure 5.12 shows what each dual augmentor might look like once the edge (u_1, u_2) (similarly (v_1, v_2)) is removed. The graph C is simply the joining of two of these objects along the two "connector" vertices (marked with empty nodes).

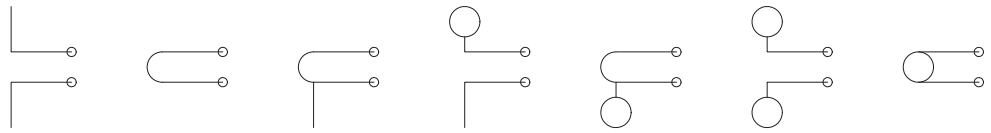


Figure 5.12: Dual augmentors with an edge removed. From left to right these are: Type-1a, Type-0, two versions of Type-1b (depending which edge is removed), two versions of Type-1c, and Type-2.

It is easy to see that all possible ways to join these objects to form C have already been proven to have a valid augmentor sum in Lemma 5.4.4, with the exception of the graph formed when the two dual augmentors A_1 and A_2 are of Type-2. Figure 5.13 shows the valid augmentor sum for this graph, which completes the proof of the lemma. □

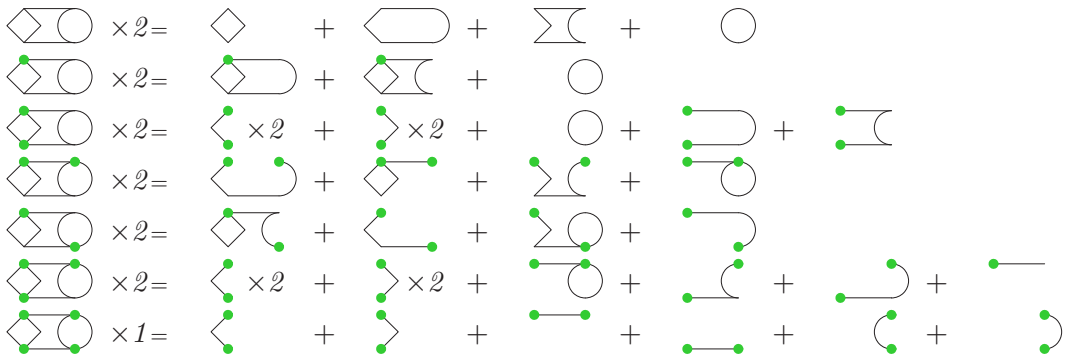


Figure 5.13: Proof that the last remaining case of Lemma 5.4.5 has a valid augmentor sum. Small circles indicate nodes of M, so this shows all possible cases.

Using the above lemmata, we are now able to prove that our algorithm works correctly.

Proof of Theorem 5.3.4. We prove this by induction on the number of nodes in $\overline{S}$. The smallest cubic multigraph consists of two nodes u, v and the entire graph is a dual augmentor of Type-2.

Now assume that all cubic multigraphs smaller than $\overline{S}$ have a valid augmentor sum. If $\overline{S}$ is not connected consider each component separately and use the inductive hypothesis. By Lemma 5.4.4, we have proved the inductive step for S containing a bridge, so we can assume S is 2-connected. If $\overline{S}$ is 3-connected, we can use Lemma 5.4.3, Lemma 5.4.6, or Lemma 5.4.7. Otherwise, we can use Lemma 5.4.5 together with the inductive hypothesis. This finishes the proof.

5.4.1 Special Case: $|M| = 1$

This section is devoted to the special case where $|M| = 1$ and $\overline{S}$ is 3-connected.

Lemma 5.4.6 *Assume that all cubic multigraphs smaller than $\overline{S}$ have valid augmentor sums, and that there exists a cut of size 3 with more than a single node on each side. If $\overline{S}$ is 3-connected and $|M| = 1$ then there there exists a valid augmentor sum of $\overline{S}$.*

Proof: This proof is similar to the proofs of Lemma 5.4.4 and Lemma 5.4.5. Let the cut of size 3 be $\{e_1, e_2, e_3\}$, and let $M = \{r\}$. Form two new 3-connected cubic multigraphs S_1, S_2 by contracting each side of the cut into a single node, with $r \in S_1$. S_1 is a smaller set with a single node of M, so by our assumption, there exists a valid augmentor sum α_1 of S_1. Let v be the node of S_2 representing the contracted side of the cut. Here we have a choice: do we say that v is in M or not? Call the first set S_2, and the second set S_2'. In the first case, S_2 is a set with a single node of M, which must have some valid augmentor sum α_2 with cover number y. In the second case, S_2' would have no nodes in M, so by Lemma 5.4.3, we also have a valid augmentor sum α_2', with cover number y'.

We can say something more specific about α_2 and α_2'. Since S_2' is a 3-connected set with no nodes in M, by the proof of Lemma 5.4.3 we can assume that α_2' is a cycle cover (i.e.,

the only dual augmentors appearing in it with positive weight are cycles). In particular, all dual augmentors containing v in α_2' contain exactly two edges adjacent to v. On the other hand, S_2 is a 3-connected set with a single node v in M. By the proof of Lemma 5.4.7, we can inductively show that all dual augmentors containing v in α_2 contain all three edges adjacent to it.

Now let w be the node of S_1 representing the contracted side of the cut. Let a be the number of times w appears in a dual augmentor of α_1 containing all 3 edges of w, and b be the number of times w appears in such a dual augmentor of α_1 containing two edges of w. Since $w \notin M$, these are the only options, and so the cover number of α_1 is $a + 2b/3$.

The idea is that we are going to attach together α_2 with the a dual augmentors above, and α_2' with the other b dual augmentors. To do this, let x be the least common multiple of y and $3y'/2$, and form new augmentor sums $x\alpha_1$, $\frac{xa}{y}\alpha_2$, and $\frac{2xb}{3y'}\alpha_2'$. This means that we now have xb dual augmentors containing exactly two edges of w in S_1, and xb dual augmentors containing exactly two edges of v in S_2', the latter coming from $\frac{2xb}{3y'}\alpha_2'$. Just as in Lemma 5.4.4 and Lemma 5.4.5, we can pair up these dual augmentors into pairs (A_1, A_2). Furthermore, since dual augmentors covering exactly two edges of a node must appear in triples (so that all edges are covered the same number of times), we can make sure that in every pair (A_1, A_2), both A_1 and A_2 use the same two edges from the set $\{e_1, e_2, e_3\}$. We can then form a multigraph C in $\overline{S}$ by patching A_1 and A_2 together, i.e., C consists of edges in $\overline{S}$ corresponding to either A_1 or A_2. Since all dual augmentors in α_2' are cycles, C must be a dual augmentor, since patching a dual augmentor in this manner together with a cycle results in a dual augmentor again.

We now consider the xa dual augmentors containing exactly three edges of w in S_1, and xa dual augmentors containing exactly three edges of v in S_2, the latter coming from $\frac{xa}{y}\alpha_2$. We pair them up in the same way, and patch them together to form subgraphs $C = A_1 \cup A_2$ for each pair (A_1, A_2). All structures C that can be formed in this way are dual augmentors.

By taking the above subgraphs C, together with the dual augmentors of α_1, α_2, and α_2' that do not intersect v or w, we form a valid augmentor sum for $\overline{S}$. □

The most difficult subcase of Theorem 5.3.4 is proven in the following main lemma of this section, which requires different techniques from most of our other proofs. We prove the cases when $\overline{S}$ is planar and non-planar separately. For the non-planar case, we show that there must be some subdivision of $K_{3,3}$ containing the node of M, and from this we form a valid augmentor sum using Theorem 5.4.2. In the planar case, we use powerful edge-coloring results.

Lemma 5.4.7 *Assume that $\overline{S}$ does not have a cut of size 3 with more than a single node on each side. If $\overline{S}$ is 3-connected and $|M| = 1$ then there exists a valid augmentor sum of $\overline{S}$.*

Proof: Let $M = \{r\}$, the three nodes adjacent to r be v_1, v_2, v_3, and denote the edge (r, v_i) by e_i. First, we address the non-planar case, the proof of which is due largely to Paul Seymour.

Non-planar Assume that $\overline{S}$ is not planar. We want to show that there exists a subdivision of $K_{3,3}$ in $\overline{S}$ with r as one of the degree-3 nodes in this $K_{3,3}$. From Theorem 2.4 in [65], it is easy to derive that this must hold. Specifically, delete r and its adjacent edges from $\overline{S}$ to form a new graph S'. Using the notation of [65], let $\{v_1, v_2, v_3\}$ be the set of "special" nodes $\overline{\Omega}$, and let the permutation Ω be (v_1, v_2, v_3). The society (S', Ω) is 3-connected, since if there were a separation of size 2, then there would be an edge cut (since S' is at most cubic, and v_1, v_2, v_3 have degree 2 in S') of size 2 separating r from some node in $\overline{S}$, which contradicts $\overline{S}$ being 3-connected. We also need to show that S' is not "rural", which means that it can be drawn in the plane without edge intersections, and so that the nodes of Ω are on the outside face. S' is not rural, since if it were, we could attach r again and make $\overline{S}$ become planar. S' cannot have a cross because $|\overline{\Omega}| = 3$. Therefore, by Theorem 2.4 in [65], S' must have a tripod. A tripod combined with r and e_1, e_2, e_3 gives us exactly a subdivision of $K_{3,3}$.

Let K be this $K_{3,3}$ instance, and let $\sigma(K)$ be the subdivision of it found above. By "a subdivision" we mean that $\sigma(K)$ is obtained from K by adding nodes in the middle of

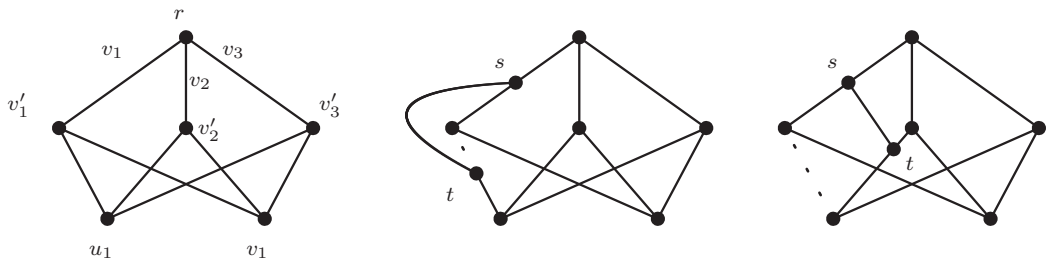


Figure 5.14: Proof of Lemma 5.4.7 (Left) A subdivision of $K_{3,3}$ (Middle) An s-t path with $t \in P(u_1, v_1')$ (Right) An s-t path with $t \in P(u_1, v_2')$

edges. As pictured in Figure 5.14, let v_1', v_2', v_3' be the nodes adjacent to r in K, with v_i' appearing at the end of a path in $\sigma(K)$ starting at v_i, and let u_1, u_2 be the nodes in K that are at distance 2 from r. For any u, v, define $P(u, v)$ to be the unique path in $\sigma(K)$ that does not go through any degree 3 node except at its endpoints. Now we want to show that there exists such a $\sigma(K)$ with the length of $P(r, v_i')$ equal to 1 for all i. Take a $\sigma(K)$ as above so that the lengths of $P(r, v_i')$ are minimal. Suppose that there exists some path P from s to t, disjoint from $\sigma(K)$ except at endpoints, with $s \in P(r, v_1')$. If $t \in P(v_1', u_1)$, then we can form a new $K_{3,3}$ instance by replacing $P(t, v_1')$ with P, as in Figure 5.14(Middle). This shortens the length of $P(r, v_1')$, since s becomes the "new" v_1', giving a contradiction. If $t \in P(u_1, v_2')$, we can similarly re-design K by replacing $P(u_1, v_1')$ with P, and making t the "new" u_1, as in Figure 5.14(Right). This also shortens $P(r, v_1')$, and all other cases can be reduced to these, so we can assume that all paths from nodes in $P(r, v_i')$ must pass through v_1', v_2', v_3' to reach nodes outside of $\cup_i P(r, v_i')$. However, if $v_i \neq v_i'$ for all i, then the cut consisting of the closest edges to v_i' on each $P(r, v_i')$ disconnects the graph with more than one node on each side, which we assumed is impossible. This gives us that $v_i = v_i'$, as desired.

Take this $\sigma(K)$, which is really the union (although not a valid sum) of two dual augmentors containing r: one with degree 3 at u_1, and one with degree 3 at u_2; both with degree 3 at r. We will now form a valid augmentor sum α for $\overline{S}$ with the cover number of $2x$. Set the α value of each of these dual augmentors to some value x. To complete this

augmentor sum, we need to cover all edges not in $\sigma(K)$ by $2x$ dual augmentors, and all edges in $\sigma(K)$ not adjacent to r by x. We do not need to cover the edges adjacent to r anymore, since we already covered them with $2x$, so we may as well remove them. This gives us three nodes of degree 2. Suppose v is such a node, with incident edges (v, w_1) and (v, w_2). v is not in M, so any dual augmentor must contain both (v, w_1) and (v, w_2). Without loss of generality, we can replace such nodes v and both incident edges with a single edge (w_1, w_2). This results in a cubic graph with edges that we need to cover either x or $2x$ times. By Theorem 5.4.2, we know we can cover this graph with cycles in the desired manner if it is 3-connected. If it were not, this would mean that there is some cut in this graph consisting of only two edges f_1, f_2. This graph was constructed by removing r, and then getting rid of the remaining degree 2 nodes, so this means that removing two edges f_1, f_2 from $\overline{S}$ together with r disconnects $\overline{S}$. Since r is degree 3, this implies that removing three edges from $\overline{S}$ disconnects it. By our assumption, the only way this is possible is if one side of the cut consists of a single node v that is adjacent to f_1, f_2, and r. However, we contracted all nodes adjacent to r, so $\{f_1, f_2\}$ could not be a cut in the resulting graph. Therefore there exists a circuit sum of this graph, giving us a valid augmentor sum together with the the augmentors in $\sigma(K)$.

Planar We now address the planar case. Tait [71] showed that the Four Color Theorem [9, 66] is equivalent to the following statement: "Every 2-connected cubic planar graph is edge-3-colorable," and more recently [19] proved this statement without relying on the Four Color Theorem. Take such a coloring of $\overline{S}$, where each color just forms a perfect matching. Call these matchings M_1, M_2, M_3 and form symmetric differences $M_1 \oplus M_2, M_2 \oplus M_3$, and $M_3 \oplus M_1$. Each of these is a set of disjoint cycles, with every edge being in exactly two of these. Consider what $M_1 \oplus M_2$ looks like with respect to r, and let C_1 be the cycle of it containing r, and C_2 be the cycle of it containing the node adjacent to r attached by an edge e of M_3. Form a dual augmentor by taking $C_1 \cup C_2 \cup \{e\}$ (note that C_1 may equal C_2). Figure 5.15 shows what this object can look like. To check that this is a dual augmentor, notice that the only nodes of degree 3 are the endpoints of e, since C_1 and C_2 are disjoint.

The only node of M is r, which has degree 3, as desired.

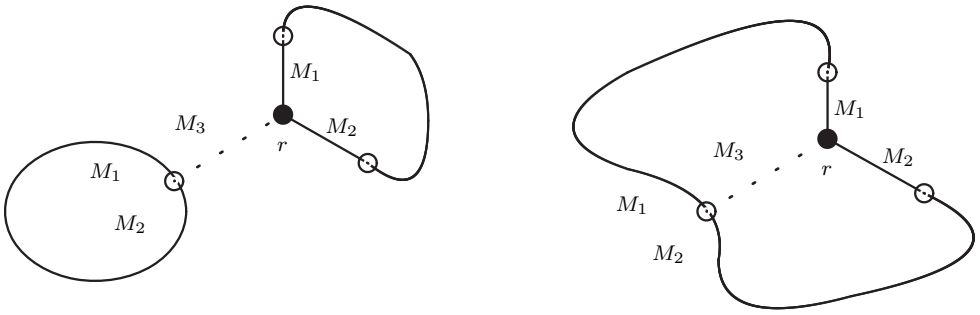


Figure 5.15: The planar case in Lemma 5.4.7. (Left) $C_1 \neq C_2$ (Right) $C_1 = C_2$

Now, take all the cycles of $M_1 \oplus M_2, M_2 \oplus M_3, M_3 \oplus M_1$, but replace C_1, C_2 by $C_1 \cup C_2 \cup \{e\}$ (and similarly for $M_2 \oplus M_3$ and $M_3 \oplus M_1$). If we take all of these dual augmentors and cycles, we get a dual augmentor cover that covers the edges next to r three times and all the other edges twice. Remove r and its adjacent edges. As in the non-planar case, we can use Theorem 5.4.2 to show that we can cover the resulting graph with cycles so that every edge appears in exactly the same number x of cycles. Together this gives a valid augmentor sum, since we can multiply the cover above by x, combine it with the cycle cover, and end up with a valid augmentor sum of size $3x$. □

5.5 Running Time

In this section we show that our algorithm for *Simplex Matching* runs in polynomial time. Recall that the algorithm is simply:

```
Start with any perfect matching.
Repeat until done
  Find an alternating 2-factor using Lemma 5.2.1.
  Augment the current matching by the above 2-factor.
```

We can find the initial perfect matching using the algorithm for unweighted *Simplex Matching* presented in Section 5.1 or modify the $\{K_2, K_3\}$ packing algorithm from [48]. If we are applying *Simplex Matching* to solve *Terminal Backup* or similar problems, then there always exists a perfect matching without 3d edges, which we can find using traditional matching algorithms. We need to estimate how much time it takes to find an alternating 2-factor according to Lemma 5.2.1 and how many times we need to repeat this step until the optimal solution is reached given that we start from some arbitrary perfect matching. We show that

Theorem 5.5.1 *Our algorithm solves* Simplex Matching *with integer edge costs in polynomial time.*

Proof: Let OPT be the cost of the best perfect matching M^*, and let M be some perfect matching. As mentioned before, $M^* \oplus M$ is an M-alternating 2-factor with at most n vertices, wherer n is the number of vertices of G. By Lemma 5.3.3 and Theorem 5.3.4 we know that there exists an augmentor A with $\phi_M(A) \geq \frac{\phi_M(M^* \oplus M)}{n}$. By Lemma 5.2.1 we can efficiently find an M-alternating 2-factor S such that $\phi_M(S) \geq \phi_M(A)$. Since $\phi_M(M^* \oplus M) = cost(M) - OPT$, then every time we augment in our algorithm, we decrease the cost by at least $\frac{cost(M)-OPT}{n}$.

Suppose that the initial matching we find is α times more expensive than OPT. If D denotes the ratio of the maximum edge cost to the minimum edge cost then clearly $\alpha \leq D$. During the first iteration we will decrease the cost of the matching by at least $\frac{\alpha-1}{n} OPT$, resulting in a matching of cost at most $(\alpha - 1)\left(\frac{n-1}{n}\right) + 1$ more expensive than OPT. After the second iteration the cost of the new matching will be at most a factor of $(\alpha-1)\left(\frac{n-1}{n}\right)^2+1$ more expensive than OPT. It is easy to see that after the k-th step the cost of the current matching would be at most a factor of $(\alpha - 1)\left(\frac{n-1}{n}\right)^k + 1$ more expensive than OPT.

The algorithm terminates when Lemma 5.2.1 returns an alternating 2-factor that does not improve the current matching. We claim that the current matching at this point has cost OPT. Suppose this is not the case, then there exists an alternating 2-factor of positive potential. By Lemma 5.3.3, there exists an augmentor with positive potential and since all

edge costs are integers this potential is at least equal to 1. By Lemma 5.2.1 we can find an alternating 2-factor with a potential at least 1 that improves the current matching and we reach the desired contradiction.

Suppose that the algorithm terminated after $k+1$ invocations of Lemma 5.2.1, the last invocation being the one that returned an alternating 2-factor that did not improve the matching. Since the edge costs are integral, it is easy to see that $k \leq t$ for any t is such that

$$\left((\alpha-1)\left(\frac{n-1}{n}\right)^t + 1\right) OPT - OPT < 1$$

$$\Leftrightarrow$$

$$t > \frac{\ln(\alpha-1) + \ln OPT}{\ln \frac{n}{n-1}}$$

Given that $\alpha \leq D$ and $\ln \frac{n}{n-1} > \frac{1}{n}$, the *Simplex Matching* algorithm executes no more than $n(\ln D + \ln OPT)$ invocations of Lemma 5.2.1. We can observe also that OPT can be bounded by nD so the number of invocations of Lemma 5.2.1 is $O(n \ln D + n \ln n)$.

Done in a naive manner, each invocation of Lemma 5.2.1 consists of running a minimum cost weighted matching algorithm for every pair of 3d edges. This could take as long as $O(n^3m^2)$, where m is the number of 3d edges. However, there are some simple ways to make this step run faster. We could take advantage of the fact that the minimum cost matchings that we are calculating are extremely related. If we use an "augmenting path" algorithm for calculating minimum cost matchings [30], then each calculation only takes $O(n^2)$ time, resulting in a running time of $O(n^3 + n^2m^2)$ for each invocation of Lemma 5.2.1. The running time could be reduced further in the geometric setting. Finally, notice that not all pairs of 3d edges need to be considered. A lot of the pairs can be eliminated in advance, significantly reducing the running time. This is especially true when applying the *Simplex Matching* algorithm to the *Terminal Backup* problem, or to any problem involving covering instead of exact matching. For more details on optimizing the running time, see Xu et al. [74].

Recall that in the first step of the algorithm we find a perfect matching in time $O(n^3m^2)$

which is dominated by the running time of the invocations of Lemma 5.2.1. Since a perfect matching exists we can assume that m is $\Omega(n)$. Thus the resulting running time of our algorithm for weighted *Simplex Matching* is $O(n^3m^2 \ln D + n^3m^2 \ln n))$. □

In the case where the edge costs are not integer, the running time will depend on how these costs are represented. By running the augmentation algorithm until the improvement is at most ε, we can obtain an algorithm that finds a solution that costs at most $OPT + \varepsilon$ in time polynomial in $\ln D$, $\ln(1/\varepsilon)$ and n.

Chapter 6

Conclusion

We studied the *Terminal Backup* problem and showed how to reduce it to *Simplex Matching*, a minimum cost perfect matching problem on hypergraphs with weighted edges of size 2 and 3 that satisfy the *Simplex Condition*. In the previous chapter we provided a polynomial time algorithm for solving *Simplex Matching*. Here we present another natural problem, *Project Assignment*, that is solvable in polynomial time via a reduction to *Simplex Matching*. We also discuss future directions of research related to *Terminal Backup* and *Simplex Matching*.

6.1 *Project Assignment*

Consider a likely task a teacher might have to perform at the end of the semester – assigning final projects to students. The teacher has a list of possible projects as well as data from each of the students indicating how happy they will be to work on any one of the projects. The goal is to assign each student to a project such that the total student "happiness" is maximized and no student works alone.

More specifically, suppose that there is a utility function $u(s, p)$ that indicates how much a student s would like working on project p. We need to break the students into groups of at least 2 and assign a project to each group such that the total sum of the students' utilities is maximized. This problem is a special case of facility location with lower bounds and can be reduced to *Simplex Matching*. Notice that if the students were allowed to work alone,

i.e. the groups can be of size 1, then the optimum would simply assign each student to the project she likes best. If the groups need to be of size exactly 2, the problem is reducible to non-bipartite matching (notice that not all projects need to be assigned, otherwise this would be easily solvable by a flow argument). And if the group size had to be at least 3, the problem becomes NP-Hard. The variant with group size at least 2, however, is reducible to *Simplex Matching* and so has nontrivial structure that can be exploited to form an efficient algorithm.

The reduction from *Project Assignment* to *Simplex Matching* is simple. Notice that if the optimal solution forms groups of size larger than 3 then there is an equivalent solution with groups of only 2 or 3 students. We form a hypergraph H with vertices corresponding to the students and edges of size 2 or 3. For every two students s_1 and s_2 form an edge (s_1, s_2) of weight equal to the smallest cost, expressed as a negation of utility, of assigning students s_1 and s_2 to a project together. For every three students s_1, s_2 and s_3 form an edge (s_1, s_2, s_3) of weight equal to the smallest cost of assigning s_1, s_2 and s_3 to a project together.

In the above hypergraph H all 2d edges (s_1, s_2) exist and if an edge (s_1, s_2, s_3) corresponds to a project p then its cost $c(s_1, s_2, s_3)$ is equal to $-u(s_1, p) - u(s_2, p) - u(s_3, p)$. Moreover, the inequality $c(s_i, s_j) \leq -u(s_i, p) - u(s_j, p)$ holds for any pair of students (s_i, s_j). Summing up for all pairs of students (s_1, s_2), (s_2, s_3) and (s_1, s_3) gives us the inequality portion of the *Simplex Condition*. Finding the optimal solution to the *Project Assignment* problem is now reduced to solving *Simplex Matching* on H.

6.2 Future Directions

There are several natural directions to pursue that are suggested by our work. First of all, the polynomial time algorithm for *Simplex Matching* is not strongly polynomial as its running time depends on the ratio of the highest to the lowest cost in the hypergraph. It would be interesting to remove that dependence.

In Theorem 5.3.4 we show that every cubic graph can be covered by dual augmentors that

are simple combinatorial objects. Our hope is that our covering results and the techniques used to obtain them will lead to more covering results where nodes have general degree constraints.

Simplex Matching seems well suited to solve problems where some elements need to be split up in groups of 2 or 3 and the problem can be made to satisfy the *Simplex Condition*. *Terminal Backup* and *Project Assignment* are two seemingly different problems of this kind. Recently Williams [73] has used the weighted *Simplex Matching* algorithm to show that the 2-*anonymity* optimization problem from data privacy is solvable in polynomial time (for definition of k-*anonymity* see [70], for discussion of its complexity see [61], for approximation algorithms see [3]). It appears that the ability to solve *Simplex Matching* efficiently is a useful tool, so we can ask what other interesting problems could be solved via a reduction to *Simplex Matching* or using the polynomial time algorithm for *Simplex Matching* as a subroutine.

Finally, *Terminal Backup* seems very similar to a large number of network design problems, yet it is one of the few such problems that can be solved in polynomial time. The polynomial time algorithm for *Terminal Backup* could provide us with an additional tool when designing improved approximation algorithms for some harder network design problems.

References

[1] Index to BellSouth FCC Tariff 1 Sections.
`http://cpr.bst.bellsouth.com/pdf/fcc/fcc1.htm`, Section 7, 1996.

[2] Z. Abrams, A. Meyerson, K. Munagala and S. Plotkin. On the Integrality Gap of Capacitated Facility Location. Techincal Report, CMU-CS-02-199, 2002.

[3] G. Aggarwal, T. Feder, K. Kenthapadi, R. Motwani, R. Panigrahy, D. Thomas and A. Zhu. Approximation Algorithms for k-Anonymity. In *Journal of Privacy Technology*, 20051120001, 2005.

[4] N. Alon, S. Hoory and N. Linial. The Moore Bound for Irregular Graphs. In *Graphs and Combinatorics*, vol. 18, pages 53–57, 2002.

[5] M. Andrews. Hardness of Buy-at-Bulk Network Design. In *Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science*, pages 115–124, 2004.

[6] M. Andrews and L. Zhang. Approximation Algorithms for Access Network Design. In *Algorithmica*, vol. 34, no. 2, pages 197–215, 2002.

[7] M. Andrews and L. Zhang. Bounds on Fiber Minimization in Optical Networks. In *Proceedings of the 24th IEEE INFOCOM*, pages 409-419, 2005.

[8] E. Anshelevich and A. Karagiozova. Terminal Backup, 3D Matching, and Covering Cubic Graphs. In *Proceedings of the 39th Annual ACM Symposium on Theory of Computing*, pages 391–400, 2007.

[9] K. Appel, W. Haken. Every Planar Map is Four-Colorable. Contemporary Mathematics, vol. 98, 1989.

[10] E. Arkin, R. Hassin, S. Rubinstein, M. Sviridenko. Approximations for Maximum Transportation Problem with Permutable Supply Vector and Others Capacitated Star Packing Problems. In *Algorithmica*, vol. 39, pages 175–187, 2004.

[11] B. Awerbuch, Y. Azar and Y. Bartal. On-line Generalized Steiner Problem. In *Proceedings of the 7th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 68–74, 1996.

[12] B.Awerbuch and Y. Azar. Buy-At-Bulk Network Design. In *Proceeding of the 38th Annual IEEE Symposium on Foundations of Computer Science*, pages 542–547, 1997.

[13] M. Balinski. Fixed-Cost Transportation Problems. In *Naval Research Logistics Quarterly*, vol. 8, pages 41–54, 1961.

[14] Y. Bartal. Probabilistic Approximation of Metric Spaces and its Algorithmic Applications. In *Proceedings of the 37th Annual IEEE Symposium on Foundations of Computer Science*, pages 184-193, 1996.

[15] Y. Bartal. On Approximating Arbitrary Metrics by Tree Metrics. In *Proceedings of the 30th Annual ACM Symposium on Theory of Computing*, pages 183–193, 1998.

[16] Y. Bartal, M. Charikar and P. Indyk. On Page Migration and Other Relaxed Task Systems. In *Proceedings of the 8th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 43–52, 1997.

[17] M. Bläser, B. Manthey. Two Approximation Algorithms for 3-Cycle Covers. In *Proceedings of the 5th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems*, pages 40–50, 2002.

[18] M. Bläser, L. Ram, M. Sviridenko. Improved Approximation Algorithms for Metric Maximum ATSP and Maximum 3-Cycle Cover Problems. In *Proceedings of the 9th*

Workshop on Algorithms and Data Structures, vol. 3608 of Lecture Notes in Computer Science, pages 350–359, 2005.

[19] I. Cahit. Spiral Chains: The Proofs of Tait's and Tutte's Three-Edge-Coloring Conjectures. arXiv preprint, math CO/0507127 v1, July 6, 2005.

[20] G. Calinescu, A. Zelikovsky. The Polymatroid Steiner Problems. In *Journal of Combinatorial Optimization*, vol. 9, no. 3, pages 281–294, 2005.

[21] M. Charikar and A. Karagiozova. On Non-Uniform Multicommodity Buy-at-Bulk Network Design. In *Proceedings of the 37th Annual ACM Symposium on Theory of Computing*, pages 176–182, 2005.

[22] C. Chekuri, M. Hajiaghayi, G, Kortsarz and M. Salavatipour. Approximation Algorithms for Non-Uniform Buy-at-Bulk Network Design. In *Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science*, pages 677–686, 2006.

[23] C. Chekuri, S. Khanna and J. Naor. A Deterministic Algorithm for the Cost-Distance Problem. In *Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 232–233, 2001.

[24] J. Chen, S. Lu, S. Sze, F. Zhang. Improved Algorithms for Path, Matching, and Packing Problems. In *Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 298-307, 2007.

[25] J. Chuzhoy, A. Gupta, J. Naor and A. Sinha. On the Approximability of Some Network Design Problems. In *Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 943–951, 2005.

[26] G. Cornuéjols. Combinatorial Optimization: Packing and Covering. CBMS-NSF Regional Conference Series in Applied Mathematics, vol. 74, SIAM 2001.

[27] G. Cornuéjols, D. Hartvigsen. An Extenstion of Matching Theory. In *Journal of Combinatorial Theory*, Series B, vol. 40, pages 285–296, 1986.

[28] G. Cornuéjols, D. Hartvigsen, W. Pulleyblank. Packing Subgraphs in a Graph. In *Operations Research Letters*, vol. 1, pages 139–143, 1982.

[29] W. Cunningham. Matching, Matroids, and Extensions. In *Mathematical Programming*, Series B, vol. 91, pages 515–542, 2002.

[30] U. Derigs. A Shortest Augmenting Path Method for Solving Minimal Perfect Matching Problems. In *Networks*, vol. 11, pages 379-390, 1981.

[31] J. Edmonds. Paths, Trees and Flowers. In *Canadian Journal of Mathematics*, no. 17, pages 449-467, 1965.

[32] M. Elkin, Y. Emek, D. Spielman and S. Teng. Lower Stretch Spanning Trees. In *Proceedings of the 37th Annual ACM Symposium on Theory of Computing*, pages 494–503, 2005.

[33] G. Even, G. Kortsarz, W. Slany. On Network Design Problems: Fixed Cost Flows and the Covering Steiner Problem. In *ACM Transactions on Algorithms*, vol. 1, no. 1, pages 74–101, 2005.

[34] J. Fakcharoenphol, S. Rao and K. Talwar. A Tight Bound on Approximating Arbitrary Metrics by Tree Metrics. In *Proceedings of the 35th Annual ACM Symposium on Theory of Computing*, pages 448–455, 2003.

[35] G. Gallo, S. Sandi and C. Sodini. An Algorithm for the Min Concave Cost Flow Problem. In *European Journal of Operational Research*, vol. 4, pages 248–255, 1980.

[36] N. Garg, R. Khandekar, G. Konjevod, R. Ravi, F. Salman and A. Sinha. On the Integrality Gap of a Natural Formulation of the Single-Sink Buy-at-Bulk Network Design Problem. In *Proceedings of the 8th Conference on Integer Programming and Combinatorial Optimization*, pages 170–184, 2001.

[37] M. Goemans, D. Williamson. A General Approximation Technique for Constrained Forest Problems. In *SIAM Journal on Computing*, vol. 24, no. 2, pages 296–317, 1995.

[38] M. Goldstein and B. Rothfarb. The One Terminal Telpak Problem. In *Operations Research*, vol. 19, pages 156–169, 1971.

[39] P. Gray. Exact Solutions of the Fixed Charge Transportation Problem. In *Operations Research*, vol. 19, pages 1529–1538, 1971.

[40] S. Guha, A. Meyerson and K. Munagala. Hierarchical Placement and Network Design Problems. In *Proceedings of the 41st Annual IEEE Symposium on Foundations of Computer Science*, pages 603–612, 2000.

[41] S. Guha, A. Meyerson and K. Munagala. A Constant Factor Approximation for the Single Sink Edge Installation Problems. In *Proceedings of the 33rd Annual ACM Symposium on the Theory of Computing*, pages 383–388, 2001.

[42] A. Gupta, A. Kumar, M. Pàl and T. Roughgarden. Approximation Via Cost-Sharing: A Simple Approximation Algorithm for the Multicommodity Rent-or-Buy Problem. In *Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science*, pages 606–615, 2003.

[43] A. Gupta, A. Kumar and T. Roughgarden. Simpler and Better Approximation Algorithms for Network Design. In *Proceedings of the 35th Annual ACM Symposium on Theory of Computing*, pages 365–372, 2003.

[44] A. Gupta, A. Srinivasan. An Improved Approximation Ratio for the Covering Steiner Problem. In *Theory of Computing*, vol. 2, pages 53–64, 2006.

[45] M. Hajiaghayi, G. Kortsarz and M. Salavatiour. Approximating Buy-at-Bulk and Shallow-Light k-Steiner Trees. In *Proceedings of the 9th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems*, LNCS 4110, pages 153–163, 2006.

[46] D. Hartvigsen, P. Hell, J. Szabó. The k-Piece Packing Problem. In *Journal of Graph Theory*, vol. 52, pages 267–293, 2006.

[47] P. Hell. Graph Packings. In *Electronic Notes in Discrete Mathematics*, vol. 5, pages 170-173, 2000.

[48] P. Hell, D. Kirkpatrick. Packings by Cliques and by Finite Families of Graphs. In *Discrete Mathematics*, vol. 49, pages 45–49, 1984.

[49] U. Jamshy, M. Tarsi. Short Cycle Covers and the Cycle Double Cover Conjecture. In *Journal of Combinatorial Theory*, Series B, vol. 56, pages 197-204, 1992.

[50] H. Kaplan, M. Lewenstein, N. Shafrir, M. Sviridenko. Approximation Algorithms for Asymmetric TSP by Decomposing Directed Regular Multigraphs. In *Journal of the ACM*, vol. 52, pages 602-626, 2005.

[51] D. Karger, M. Minkoff. Building Steiner Trees with Incomplete Global Knowledge. In *Proceedings of the 41st Annual IEEE Symposium on Foundations of Computer Science*, pages 613-623, 2000.

[52] A. Kelmans. Optimal Packing of Induced Stars in a Graph. In *Discrete Mathematics*, vol. 173, pages 97–127, 1997.

[53] A. Kumar, A. Gupta and T. Roughgarden. A Constant Factor Approximation Algorithm for the Multicommodity Rent-or-Buy Problem. In *Proceedings of the 43rd Annual IEEE Symposium on Foundations of Computer Science*, pages 333–344, 2002.

[54] M. Loebl, S. Poljak. Efficient Subgraph Packing. In *Journal of Combinatorial Theory*, Series B, vol. 59, pages 106–121, 1993.

[55] L. Lovasz, M. Plummer. Matching Theory. Elsevier Science Ltd, 1986.

[56] G. Lueker. Two NP-Complete Problems in Non-Negative Integer Programming. Technical report, Princeton University, Princeton, NJ, CS-178, 1975.

[57] Y. Mansour and D. Peleg. An Approximation Algorithm for Minimum-Cost Network Design. In *Proceedings of the DIMACS Workshop on Robust Communication Networks: Interconnection and Survivability*, DIMACS series vol. 53, pages 97–106, 1998.

[58] A. Meyerson. Online Algorithms for Network Design. In *Proceedings of the 16th ACM Symposium on Parallelism in Algorithms and Architectures*, pages 275–280, 2004.

[59] A. Meyerson, K. Munagala and S. Plotkin. Cost-Distance: Two-Metric Network Design. In *Proceedings of the 41st Annual IEEE Symposium on Foundations of Computer Science*, pages 624–630, 2000.

[60] A. Meyerson, K. Munagala and S. Plotkin. Designing Networks Incrementally. In *Proceedings of the 42nd Annual IEEE Symposium on Foundations of Computer Science*, pages 406–415, 2001.

[61] A. Meyerson and R. Williams. General k-anonymization is Hard. *CMU Technical Report CMU-CS-03-113*, 2003.

[62] P. Mirchandani. Polyhedral Structure of a Capcitated Network Design Problem with an Application to the Telecommunications Industry. Ph.D. Thesis, Sloan School of Management, MIT, 1989.

[63] G. Pap. Hypo-Matchings in Directed Graphs. In *Graph Theory 2004*, Birkhuser Verlag, Basel, Switzerland, pages 325-335, 2006.

[64] G. Pap. A TDI Description of Restricted 2-Matching Polytopes. In *Proceedings of the 10th Conference on Integer Programming and Combinatorial Optimization*, pages 139–151, 2004.

[65] N. Robertson, P.D. Seymour. Graph Minors. IX. Disjoint Crossed Paths. In *Journal of Combinatorial Theory*, Series B, vol. 49, no. 1, pages 40–77, 1990.

[66] N. Robertson, D.P. Sanders, P.D. Seymour, R. Thomas. The Four-Colour Theorem. In *Journal of Combinatorial Theory*, Series B, vol. 70, no. 1, pages 2–44, 1997.

[67] S. Salman, J. Cheriyan, R. Ravi, S. Subramanian. Buy-at-Bulk Network Design: Approximating the Single-Sink Edge Installation Problem. In *Proceedings of the 8th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 619–628, 1997.

[68] A. Schrijver. Combinatorial Optimization: Polyhedra and Efficiency. Springer-Verlag, 2003.

[69] P.D. Seymour. Sums of Circuits. In *Graph Theory and Related Topics*. J.A. Bondy and U.R.S. Murty Eds, Academic Press, pages 341-355, 1978.

[70] L. Sweeney. k-Anonymity: A Model for Protecting Privacy. in *International Journal on Uncertainty, Fuzziness and Knowledge-based Systems*, vol. 10, no. 5, pages 557-570, 2002.

[71] P.G. Tait. Note on a Theorem in Geometry of Position. In *Transactions of the Royal Society of Edinburgh*, vol. 29, pages 657-660, 1880.

[72] K. Talwar. The Single-Sink Buy-at-Bulk LP Has Constant Integrality Gap. In *Proceedings of the 9th Conference on Integer Programming and Combinatorial Optimization*, LNCS vol 2337, pages 475–486, 2002.

[73] R. Williams, personal communication, 2007.

[74] D. Xu, E. Anshelevich, M. Chiang. Simplex Cover: Modeling, Algorithms, and Networking Applications. manuscript, `http://www.princeton.edu/~dahaixu/pub/simplex/simplex.pdf`.

[75] C.Q. Zhang. Integer Flows and Cycle Covers of Graphs. Marcel Dekker, 1997.

[76] W. Zangwill. Minimum Concave Cost Flows in Certain Netowkrs. In *Management Science*, vol. 14, no. 7, pages 429–450, 1968.