Iterative Methods in Combinatorial Optimization

Mohit Singh

Submitted in partial fulfilment of the requirements
of the degree of Doctor of Philosophy in Algorithms, Combinatorics and Optimization

May 2008

Tepper School of Business
Carnegie Mellon University

Abstract

Linear programming has been a successful tool in combinatorial optimization to achieve polynomial time algorithms for problems in $\mathcal{P}$ and also to achieve good approximation algorithms for problems which are $\mathcal{NP}$-hard. We demonstrate that *iterative methods* give a general framework to analyze linear programming formulations of combinatorial optimization problems. We show that iterative methods are well-suited for problems in $\mathcal{P}$ and lead to new proofs of integrality of linear programming formulations. We then use the new proofs as basic building blocks for obtaining approximation algorithms for various extensions which are $\mathcal{NP}$-hard.

In this thesis, we focus on degree bounded network design problems. The most studied problem in this class is the Minimum Bounded Degree Spanning Tree problem defined as follows. Given a weighted undirected graph with degree bound B, the task is to find a spanning tree of minimum cost that satisfies the degree bound. We present a polynomial time algorithm that returns a spanning tree of optimal cost and maximum degree $B+1$. This generalizes a result of Furer and Raghavachari [37] to weighted graphs, and thus settles a 15-year-old conjecture of Goemans [42] affirmatively. This is also the best possible result for the problem in polynomial time unless $\mathcal{P} = \mathcal{NP}$.

We also study more general degree bounded network design problems including the Minimum Bounded Degree Steiner Tree problem, the Minimum Bounded Degree Steiner Forest problem, the Minimum Bounded Degree k-Edge Connected subgraph problem and the Minimum Bounded Degree Arborescence problem. We show that iterative methods give bi-criteria approximation algorithms that return a solution whose cost is within a small constant multiplicative factor of the optimal solution and the degree bounds are violated by an additive error in undirected graphs and a small multiplicative factor in directed graphs. These results also imply first additive approximation algorithms for various degree constrained network design problems in undirected graphs.

We demonstrate the generality of the iterative method by applying it to the degree constrained matroid problem, the multi-criteria spanning tree problem, the multi-criteria matroid basis problem and the generalized assignment problem achieving or matching best known approximation algorithms for them.

Acknowledgements

First and foremost, I must thank my advisor R. Ravi. He has been a great source of inspiration and learning. I especially thank him for giving me the freedom to work and having faith in me even when theorems were hard to come and proofs were going wrong.

I also thank my friend and collaborator Lap Chi Lau. His style of working has been a great influence on me. It was an amazing experience to spend last two summers interacting with him in Seattle and working over the phone and email over the last couple of years.

I have been very fortunate to be a part of the ACO group here at Carnegie Mellon. I thank people I worked closely with, including Kedar Dhamdhere, Anupam Gupta, Daniel Golovin, Vineet Goyal and Viswanath Nagarajan and people I interacted with, such as Egon Balas, Gerard Cornuejols, Alan Frieze, Francois Margot, Oleg Pikhurko and Anureet Saxena.

I would also like to thank the theory group at Microsoft Research for my summer internships in summer of 2006 and 2007. I especially thank Uri Feige who mentored (and tolerated) me both the summers. It was a learning experience to just watch him work. I also thank Seffi Naor and Kamal Jain with whom I really enjoyed working. I also thank my external committee member Michel Goemans for comments on my work and the thesis.

I am very grateful to the support I received during my graduate studies including the William Larimer Mellon Fellowship, NSF grants CCR-0430751, CCF-0728841 and the Aladdin grant.

I thank my Royal Garden friends, Vineet, Deepak, Mohit, Prasad, Viswanath and Amitabh who made the stay in Pittsburgh enjoyable and full of fun. I thank my brother Gyanit, sister Tripti and Ankita who have all been a great support. Finally, I would like to thank my parents for their love and advice.

Dedicated to my parents.

Contents

List of Figures

List of Tables

1

Introduction

Combinatorial optimization deals with obtaining efficient solutions to discrete problems. In a seminal paper, Edmonds [26] advocated that an algorithm be considered efficient if the number of atomic operations the algorithm takes to return a solution is polynomial in the problem size. On the contrary, an algorithm which requires superpolynomial number of operations is not considered efficient. For example, in many cases, a brute force search can be exponential in the problem size, and hence, not considered efficient.

This leads to a natural classification of problems into *simple* and *hard* problems, depending on whether one obtains a polynomial time algorithm for them. Cook [17] formalized this classification by introducing the complexity classes $\mathcal{P}$ and $\mathcal{NP}$, and the notion of $\mathcal{NP}$-completeness. Cook [17] and Karp [55] proved that a large class of natural combinatorial optimization problems are $\mathcal{NP}$-complete and that we cannot hope to obtain polynomial time algorithms for such problems, unless $\mathcal{P} = \mathcal{NP}$. Over the years, most combinatorial optimization problems have been proven to be in $\mathcal{P}$ or are $\mathcal{NP}$-complete. Importantly, most practical problems are $\mathcal{NP}$-complete.

For many problems in $\mathcal{P}$, fast polynomial time algorithms which solve the problems optimally have been developed (see Schrijver [91]). For example, the problems of computing minimum spanning trees and matchings in graphs have polynomial time algorithms. On the other hand, one cannot obtain polynomial time exact algorithms for NP-complete problems, unless $\mathcal{P} = \mathcal{NP}$. For such problems, the focus has been to seek algorithms which try to solve these problems approximately. An important area of research is to compute bounds on the worst case performance of approximate algorithms that run in polynomial time, as compared to the optimum solutions [49, 99].

A large class of $\mathcal{NP}$-complete problems are obtained by introducing *side constraints*

to simple problems. In this thesis we demonstrate that iterative methods give a general methodology for dealing with such side constraints. First, we give new iterative proofs that natural linear programming formulations for these simple problems are integral. We then extend the integrality results to LP relaxations of $\mathcal{NP}$-hard problems to obtain approximation algorithms. We apply this framework to degree constrained network design problems and obtain (almost) optimal approximation algorithms and, in some cases, we also obtain *additive* approximation algorithms.

1.1 Definitions

Given a minimization problem, let $\mathbb{S}$ be the set of all feasible solutions. The value of the objective function for any $A \in \mathbb{S}$ is denoted by $f(A)$. Let $S^* = \text{argmin}_A f(A)$ denote an optimum solution. An algorithm is called a (multiplicative) α-approximation algorithm if

$$f(S) \leq \alpha f(S^*)$$

where S is the solution returned by the algorithm. An algorithm is called an additive α-approximation algorithm if

$$f(S) \leq f(S^*) + \alpha$$

Additive approximation algorithms are rare and have been obtained for a selected few problems, including edge-coloring [100], coloring in planar graphs [2], bounded-degree spanning and Steiner trees [37, 42].

Bi-criteria approximation algorithms We also consider problems which have two objective functions f and g and one seeks a solution which minimizes both objectives simultaneously. A standard approach is to give a bound for one of the objective functions, say $g(S) \leq B$ and seek the best solution with respect to the other objective, f. Let $S^* = argim_S\{f(S) : g(S) \leq B\}$. A multiplicative (α, β)-approximation algorithm returns a solution S such that

$$f(S) \leq \alpha f(S^*) \text{ and } g(S) \leq \beta B$$

One may also seek bi-criteria approximation algorithms where one of the objective function is violated by an additive amount and the other by a multiplicative factor.

In this thesis, we will study bounded-degree network design problems. The most

studied problem in this class is the MINIMUM BOUNDED-DEGREE SPANNING TREE (MBDST) problem which is defined as follows. Given an undirected graph with degree upper bound B_v on each vertex v and a cost function on the edges, the task is to find a spanning tree of minimum cost which satisfies all the degree bounds. Observe that we have two objective functions in the MBDST problem, degree and cost and the goal is find bicriteria approximation algorithms. The cost is usually approximated by a multiplicative factor but the degree is approximated by a combination of a multiplicative factor and an additive amount.

Degree constrained versions of more general network design problems, the STEINER TREE problem, the STEINER FOREST problem, the ARBORESCENCE problem, and the STRONGLY K-EDGE CONNECTED subgraph problem, are similarly defined. We will obtain bi-criteria approximation algorithms for the above problems which approximate both the cost and the degree of the solution.

1.2 Our Contributions and Results

In this thesis, we propose *iterative methods* as a general technique to achieve structural results for simple combinatorial optimization problems, and use this as a building block for obtaining approximation algorithms for NP-hard variants.

The first step of the iterative method is the application to underlying base problem that is polynomial time solvable and showing that a natural linear programming relaxation has integral vertex solutions. The integrality is shown in an inductive manner and gives new integrality proofs for the following problems. These results were obtained jointly with Lau and Ravi [68] and appear in Chapter 3.

Theorem 1.1 *The iterative method shows that a natural linear programming relaxations for the following problems has integral vertex solutions.*

1. *Minimum spanning tree.*
2. *Maximum weight bipartite matching.*
3. *Minimum cost arborescence.*
4. *Minimum cost base in matroids.*
5. *Minimum cost perfect matching in general graphs.*

The real power of the new proofs, obtained via the iterative method, is realized when we consider NP-hard variants of the underlying simply problems. The crucial observation is

that a large class of NP-hard problems are simple problems with additional side constraints. For example, adding degree bound constraints to the minimum spanning tree problem gives us the MINIMUM BOUNDED-DEGREE SPANNING TREE problem.

We introduce the iterative relaxation method to deal with the additional side constraints. A generic relaxation step proceeds as follows.

Iterative Relaxation. Fix a threshold β. If there is a constraint $\sum_i a_i x_i \leq b$ such that $\sum_i a_i \leq b + \beta$ then remove the constraint.

For problems where variables take values from $\{0, 1\}$, the iterative relaxation step ensures that the corresponding constraint can only be violated by an additive error of β even after it is removed. This step is crucial in achieving *additive* approximation algorithms. We also note that the iterative relaxation method is similar to the techniques of Beck and Fiala [6] result on discrepancy of sets. One application of the iterative relaxation technique gives us the following results that we obtain jointly with Lau [96] and appears in Chapter 4.

Theorem 1.2 *There exists a polynomial time algorithm for the* MINIMUM BOUNDED-DEGREE SPANNING TREE *problem which returns a tree of optimal cost such that the degree of any vertex v in the tree is at most $B_v + 1$. Here, the optimal cost is the cost of the minimum cost tree which satisfies the degree bounds exactly.*

Theorem 1.2 is the optimal result for the MBDST problem unless $\mathcal{P} = \mathcal{NP}$ and positively resolves a conjecture of Goemans [42].

Iterative rounding technique was introduced by Jain [53] for approximation general network design problems and works as follows for typical cut-covering formulations of general connectivity problems.

Iterative Rounding. Fix a threshold $\alpha \geq 1$. If there is a variable x_i which the LP sets to a value of at least $\frac{1}{\alpha}$ then pick the corresponding element in the integral solution.

We extend the work of Jain [53] and use the two steps, rounding and relaxation, in various combinations to derive strong approximation algorithms for a large class of problems.

- We obtain bi-criteria approximation algorithm for the MINIMUM BOUNDED-DEGREE STEINER TREE problem, the MINIMUM BOUNDED-DEGREE STEINER FOREST problem and the MINIMUM BOUNDED-DEGREE STEINER NETWORK problem. The solution returned by the algorithm costs at most twice the optimal solution and the

Network Topology	Single Criterion		Previous Bi-criteria	Our results
	Cost	Degree		
Spanning Tree	1	$B+1$ [37]	$(1, B+2)$ [42]	$(1, B+1)$
Steiner Tree	1.55 [90]	$B+1$ [37]	$(4, 2B+\log n)$ [61]	$(2, B+3)$
Steiner Forest	2 [1]	$B+3^*$	$(O(\log n), O(\log n))$	$(2, B+3)$
k-EC Subgraph	2 [53]	$B+O(k)^*$	$(k\log n, k\log n)$ [33]	$(2, B+O(k))$
Steiner Network	2 [53]	$B+O(r_{max})^*$	-	$(2, B+O(r_{max}))$
k-Strongly EC	2 [101]	$3B+5^*$	-	$(3, 3B+5)$
Arborescence	1	$2B+2^*$	-	$(2, 2B+2)$

Table 1.1: Results on Minimum Cost Degree-Bounded Network Design Problems, where * denotes that our results on bi-criteria approximation also improve the single-criterion guarantees.

degree of any vertex violates its degree bound by an additive amount which depends on the maximum connectivity requirement. These results were obtained jointly with Lap Chi Lau [70] improving on the previous results obtained jointly with Lau, Naor and Salavatipour [69] and are presented in Chapter 5.

- As a corollary to the previous results, we also obtain first additive approximation algorithms for the Bounded-Degree Steiner Forest problem and the Bounded-Degree k-edge connected subgraph problem for bounded k.

- We obtain constant factor bi-criteria approximation algorithm for the Minimum Bounded-Degree Arborescence problem and the Minimum k-arc connected subgraph problem where both the cost and the maximum degree of the solution is within constant multiplicative factor of the optimal solution. These results were obtained jointly with Lap Chi Lau, Seffi Naor and Mohammad Salavatipour [69] and are presented in Chapter 6.

We also show applications of iterative methods to other combinatorial optimization problems and give the following results in Chapter 7.

- We obtain a 2-approximation algorithm for the generalized assignment problem matching the result of Shmoys and Tardos [95].

- We obtain a polynomial time approximation scheme (PTAS) for the multi-criteria spanning tree problem and the multi-criteria matroid basis problem. These results were obtained jointly with R. Ravi [89].

- We obtain an additive approximation algorithm for the degree constrained matroid problem which generalizes the Minimum bounded Degree spanning tree prob-

lem and Minimum Crossing Tree problem. This result was obtained jointly with Király and Lau [58] which also contain results on degree constrained submodular flow problem that are not included in this thesis.

1.3 Related Work

Exact LP Formulations. Linear programming has been used in combinatorial optimization soon after the simplex algorithm was developed by Dantzig [23] in the 1940's. Earlier combinatorial results of König [63] and Egerváry [32] on bipartite matchings and results of Menger [75] on disjoint paths in graphs were interpreted as integrality of linear programming formulations for these problems. Many other problems like maximum flow, assignment and transportation were also shown to be solvable by formulating linear programs for these problems which are integral. Edmonds [26, 27, 28, 29] formulated linear programs for basic problems like matching, arborescence, matroids, matroid intersection and showed that the formulations are integral. Total Dual Integrality [31] and Total Unimodularity [11] were developed as general techniques to show the integrality of linear programming formulations. The *uncrossing technique*, which is used to simplify complicated set-systems while preserving certain structural properties, has played a crucial role in combinatorial optimization (see [28, 31, 35, 44] for some applications). The uncrossing technique plays an important role in the results in our work as well and appears throughout the thesis. We refer the reader to Schrijver [91, 93] and Nemhauser and Wolsey [80], Cook et al [18] for extensive historical as well as technical details for above mentioned topics.

Network Design. Much focus has been given to designing approximation algorithms for network design problems; we refer the reader to [65] for a survey. Primal-dual algorithms initially played a central role in achieving strong approximation algorithms for network design problems starting with Agarwal, Klein and Ravi [1] and Goemans and Williamson [43]. The techniques were then applied to general connectivity problems (see [41]) with moderate success. A breakthrough result is due to Jain [53] who gave a 2-approximation algorithm for the edge-connectivity Survivable Network Design problem and introduced the iterative rounding framework. This result considerably generalized and improved previous work on network design problems. Later, the iterative rounding approach was applied to other settings including network design problems in directed graphs [38, 101] and undirected graphs [14, 34].

Degree Constrained Network Design. Network design problems with degree constraints have been studied extensively lately. A simpler setting is minimizing the max-

imum degree of a subgraph (without considering the cost) satisfying certain connectivity requirements. A well-known example is the MINIMUM DEGREE SPANNING TREE (MDST) problem, where the objective is to find a spanning tree of smallest maximum degree. This problem is already NP-hard as it generalizes the HAMILTONIAN PATH problem. Fürer and Raghavachari [37] gave an elegant approximation algorithm returning a solution with maximum degree at most one more than the optimal solution developing on the work on Win [106]. Furer and Raghavachari [37] also generalized their result to the MINIMUM DEGREE STEINER TREE problem. Ravi, Raghavachari, and Klein [59, 87] considered the MINIMUM DEGREE k-EDGE-CONNECTED SUBGRAPH problem, and gave an approximation algorithm with performance ratio $O(n^{\delta})$ for any fixed $\delta > 0$ in polynomial time, and $O(\log n / \log\log n)$ in sub-exponential time. Recently, Feder, Motwani and Zhu [33] obtained a polynomial time $O(k \log n)$-approximation algorithm.

For the general problem of finding a minimum cost subgraph with given connectivity requirements and degree bounds B_v on every vertex v, the most-studied case is the MINIMUM BOUNDED-DEGREE SPANNING TREE (MBDST) problem. The first approximation was an $(O(\log n), O(\log n))$-algorithm by [86]. This was subsequently improved in a series of papers [12, 13, 60, 62, 88] using a variety of techniques which included primal dual algorithms, Lagrangian relaxation, push-preflow framework and LP rounding algorithms. Recently, Goemans [42] made a breakthrough for this problem by giving a $(1, B_v + 2)$-approximation algorithm. The algorithm in [42] uses matroid intersection and graph orientation algorithms to round a linear programming relaxation for the problem.

Very little was known for more general connectivity requirements before our work. For the MINIMUM BOUNDED-DEGREE STEINER TREE problem, there is an $(O(\log n), O(\log n))$ approximation algorithm [86]. This bound was improved to $(O(1), O(B_v + \log n))$-approximation by [61], but the algorithm runs in quasi-polynomial time.

How to Read this Thesis

The thesis can certainly be read in a linear fashion. We highlight the dependence in various sections which may help the reader to jump to their favorite part of the thesis. We discuss linear programming basics and definitions in Chapter 2 which will be used throughout the thesis. In Chapter 3 we apply the iterative framework to show integrality of linear programming relaxations for basic combinatorial optimization problems. It is recommended that the reader read Section 3.1 for a simple application of the iterative method and Section 3.2 where the uncrossing technique is illustrated.

The latter half of the thesis deals with constrained versions of the simple *base* problems in Chapter 3. Chapter 4 on the MINIMUM BOUNDED-DEGREE SPANNING TREE problem and Section 7.2 on MULTI-CRITERIA SPANNING TREE problem build on material in the spanning tree Section 3.2. Algorithms for the MULTI CRITERIA MATROID BASIS problem and the DEGREE CONSTRAINED MATROIDS in Chapter 7 develop on the iterative algorithm for matroid basis problem in Section 3.4. Algorithms for the MINIMUM BOUNDED-DEGREE ARBORESCENCE problem in Section 6.2 develops on Section 3.3 on the arborescence problem. The treatment of generalized assignment problem in Section 7.1 develops on the bipartite matching problem in Section 3.1.

In Chapter 5 we consider degree constrained versions of general network design problems in undirected graphs which develops on the work of Jain [53]. In Chapter 6 we consider degree constrained network design problems in directed graphs which develops on work of Gabow [38]. Nonetheless, we have made an effort to make these sections as much self-contained as possible.

2

Linear Programming Basics

In this chapter we discuss some linear programming basics. We also prove a crucial rank lemma which will be used throughout the thesis. We close by reviewing some standard graph and set notation used in the thesis.

2.1 Linear Programs and Vertex Solutions

Using matrix notation, a linear program is expressed as follows.

$$\begin{array}{lrcl} \text{minimize} & c^T x & & \\ \text{subject to} & Ax & \geq & b \\ & x & \geq & 0 \end{array}$$

If x satisfies $(Ax \geq b, x \geq 0)$, then x is *feasible*. If there exists a feasible solution to an LP, it is *feasible*; otherwise it is *infeasible*. An *optimal* solution x^* is a feasible solution such that $c^T x^* = \min\{c^T x \text{ s.t.} Ax \geq b, x \geq 0\}$. The LP is unbounded (from below) if $\forall \lambda \in \mathbb{R}, \exists$ feasible x such that $c^T x < \lambda$. The space of all feasible solutions $P = \{x : Ax \geq b, x \geq 0\}$ is a called a *rational* polyhedron if all entries of A and b are rational. In this thesis we work only with rational polyhedra.

There are different forms in which a general linear program can be represented including the *normal form* and the *standard form*. Simple linear transformations show that these forms are identical to the general linear program we consider above [16].

Definition 2.1 *Let $P = \{x : Ax \geq b, x \geq 0\} \subseteq \mathbb{R}^n$. Then x is a* **vertex** *of P if there does not exist $y \neq 0$ such that $x + y, x - y \in P$.*

For any polyhedron P, we define P_I to denote the convex hull of all integral vectors in P, i.e. $P_I = conv\{x : Ax \geq b, x \geq 0, x \in \mathbb{Z}^n\}$.

Definition 2.2 *P is called integral polyhedron if $P = P_I$, i.e., P is the convex hull of all integral vectors in P.*

We call a linear program $\min\{cx : x \in P\}$ integral if P is integral. The following equivalent conditions follow easily (see Schrijver [93], pages 231-232) and condition (2) is used to show the integrality of a polyhedron throughout the thesis.

Theorem 2.3 *The following conditions are equivalent.*

1. *P is integral.*
2. *$\min\{cx | x \in P\}$ is attained by an integral vector, for each c for which the minimum is finite.*

We now show basic properties about vertex solutions. Most proofs are standard and we give a short sketch. The reader is referred to Chvátal [16] or lecture notes by Goemans [40] for details. The following lemma directly implies Theorem 2.3.

Lemma 2.4 *Let $P = \{x : Ax \geq b, x \geq 0\}$ and assume that $\min\{c^T x : x \in P\}$ is finite. Then $\forall x \in P, \exists$ a vertex $x' \in P$ such that $c^T x' \leq c^T x$. i.e., there is always a vertex optimal solution.*

Proof: The idea of the proof is that by using the definition of a vertex, we can move from a current optimal solution to one that has more zero components or more tight constraints and is closer to being a vertex. Let A_i denote the i^{th} row of A and b_i denotes the i^{th}-coordinate of the vector b.

Consider x such that it is optimal but not a vertex. That implies there exists $y \neq 0$ such that $x + y \in P$ and $x - y \in P$. Therefore,

$$\begin{aligned} A(x+y) &\geq b, \quad x + y \geq 0 \\ A(x-y) &\geq b, \quad x - y \geq 0 \end{aligned}$$

Let $A^=$ be the submatrix of A restricted to the rows which are at equality at x and $b^=$ be the vector b restricted to these rows. Hence, we have $A^=x = b^=$. Hence, we must have $A^=y \geq 0$ and $A^=(-y) \geq 0$. Subtracting, we get $A^=y = 0$. Since x is optimal, the following holds,

$$\begin{aligned} c^T x &\leq c^T(x+y) \\ c^T x &\leq c^T(x-y) \\ \Rightarrow c^T y &= 0 \end{aligned}$$

Moreover, since $y \neq 0$, without loss of generality assume there exists j such that $y_j < 0$ (if not then consider $-y$). Consider $x + \lambda y$ for $\lambda > 0$ and increase λ until $x + \lambda y$ is no longer feasible (due to the non-negativity constraints). Formally, let

$$\lambda^* = \min\{\min_{j:y_j<0} \frac{x_j}{-y_j}, \min_{i:A_i x > b_i, A_i y < 0} \frac{A_i x - b_i}{-A_i y}\}$$

Now $x + \lambda^* y$ is a new optimal solution with one more zero coordinate or one extra tight constraint. Since $x + y \geq 0$ and $x - y \geq 0$, if $x_i = 0$ then $y_i = 0$. Therefore, the coordinates that were at 0, remain at 0. Moreover $A^=(x+y) = A^=x = b$ since $A^=y = 0$, hence tight constraints remain tight. Hence this process terminates with at a vertex as claimed. □

The next theorem relates vertex solutions to corresponding non-singular columns of the constraint matrix.

Lemma 2.5 *Let $P = \{x : Ax \geq b, x \geq 0\}$. For $x \in P$, let $A^=$ denote the matrix consisting of rows of A which are satisfied at equality by x and $A^=_x$ denote the submatrix of $A^=$ consisting of the columns corresponding to the nonzeros in x. Then x is a vertex iff $A^=_x$ has linearly independent columns (i.e., $A^=_x$ has full column rank).*

Proof: ($\Leftarrow$) If x is not a vertex, we will show that $A^=_x$ has linearly dependent columns. By the hypothesis, there exists $y \neq 0$ such that $A^=y = 0$ (see the proof of the previous lemma). Therefore $A^=_y$ (the columns where y has a nonzero coordinate) has linearly dependent columns. By the observation made at the end of the previous proof, $x_j = 0 \Rightarrow y_j = 0$. Therefore, $A^=_y$ is a submatrix of $A^=_x$. Therefore, the columns of $A^=_x$ are linearly dependent.

($\Rightarrow$) We want to show that if $A^=_x$ has linearly dependent columns then x is not a vertex. By the hypothesis, there exists $y \neq 0$ such that $A^=_x y = 0$. Complete y to an

n-dimensional vector by setting the remaining coordinates to 0. Now by construction, $A^{=}y = 0$. Moreover, by construction $y_j = 0$ whenever $x_j = 0$. Also note that there exists $\epsilon \neq 0$ such that $x + \epsilon y \geq 0$ and $x - \epsilon y \geq 0$. Moreover $x + \epsilon y$ and $x - \epsilon y$ are feasible since $A(x + \epsilon y) = Ax + \epsilon Ay \geq b$ and $A(x - \epsilon y) \geq b$ for small enough $\epsilon > 0$. Hence, x is not a vertex. □

We get the following Rank Lemma that will form a basic ingredient of all iterative proofs which states that each vertex solution is determined by n linearly independent constraints at equality where n is the number of variables (See Schrijver [93], page 104).

Lemma 2.6 ***(Rank Lemma)*** *Let $P = \{x : Ax \geq b, x \geq 0\}$ and let x be a vertex of P such that $x_i > 0$ for each i. Then any maximal number of linearly independent tight constraints of form $A_i x = b_i$ for some row i of A equals the number of variables.*

Proof: Since $x_i > 0$ for each i, we have $A_x^{=} = A^{=}$. From Lemma 2.5 it follows that $A^{=}$ has full column rank. Since the number of linearly independent columns equals the number of non-zero variables in x and the row rank of any matrix equals the column rank [52] we have that the row rank of $A^{=}$ equals the number of variables. Then any maximal number of linearly independent tight constraints is exactly the maximal number of linearly independent rows of $A^{=}$ which is exactly the row rank of $A^{=}$ and hence the claim follows. □

2.2 Solving Linear Programs

In this section, we briefly mention various methods of solving linear programs.

2.2.1 Simplex Algorithm

Simplex algorithm was developed in 1940's by Dantzig [23] for solving linear programs to optimality. The idea of the simplex algorithm is to move from vertex to vertex ,along edges of the polyhedron, until we reach the optimal vertex solution. An important implementation issue is that vertices and edges of the polyhedron are not specified explicitly but represented by sets of linearly independent tight constraints (basis) as given by Lemma 2.6. Unfortunately, such a representation need not be unique and leads to technical issues (degeneracy) which need to be addressed. We refer the reader to Chvátal [16] for details. Many

variants of the simplex algorithm have been considered, each defined by which neighboring vertex to move to. Although the simplex algorithm works efficiently in practice, there are examples where each variant of the simplex algorithm runs in exponential time.

2.2.2 Polynomial Time Algorithms

Polynomial time algorithms for solving linear programs fall in two categories: The ellipsoid algorithm [56] and interior point algorithms [54]. We refer the reader to Nemhauser and Wolsey [80] and Wright [108] for details about these algorithms. Both these algorithms solve linear programs to give *near* optimal solution in polynomial time. Moreover, there are rounding algorithms [80] which *convert* a *near* optimal solution to an optimal vertex solution.

Theorem 2.7 *[80] There is a polynomial time algorithm which returns an optimal vertex solution to a linear program.*

2.2.3 Separation and Optimization

In this thesis, we will encounter linear programs where the number of constraints is exponential in the size of the problem (e.g. in the spanning tree problem in Chapter 4, we will write linear programs where the number of constraints is exponential in the size of the graph) and it is not obvious that one can solve them in polynomial time. We use the notion of separation to show that sometimes exponentially sized linear programs can be solved in polynomial time.

Definition 2.8 *Given $x^* \in \mathbb{Q}^n$ and a polyhedron $P = \{x : Ax = b, x \geq 0\}$, the* **separation problem** *is the decision problem whether $x^* \in P$. The solution of the separation problem is the answer to the membership problem and in case $x^* \notin P$, it should return a valid constraint $\pi x \geq \pi_0$ for P which is violated by x^*, i.e., $\pi x^* < \pi_0$.*

The framework of ellipsoid algorithm argues that if one can solve the separation problem for P in polynomial time then one can also optimize over P in polynomial time. Theorem 2.9 of Grötschel, Lovász and Schrijver [45] showed that polynomial time separation is equivalent to polynomial time solvability of a linear program. The basis of this equivalence is the ellipsoid algorithm. We now state the equivalence formally (see Schrijver [93], Chapter 14 for details). The *size* of a rational number is defined to be

$\log(|p|+1) + \log q$ and the size of a set of inequalities $Ax \leq b$ is defined to be the sum of sizes of entries in A and b plus mn where A is a m by n matrix. Let for each $i \in \mathbb{N}$, $P(i)$ be a rational polyhedron and suppose that we can compute, for each $i \in \mathbb{N}$, in time polynomially bounded in $\log i$, the natural numbers n_i and σ_i where $P(i) \subseteq R^{n_i}$ and such that P_i has a representation of size at most σ_i. Then the separation problem for $P(i)$ for $i \in \mathbb{N}$ is said to *polynomial time solvable* if there exists an algorithm, which for input (i, y), with $i \in \mathbb{N}$ and $y \in \mathbb{Q}^{n^i}$ solves the separation problem for $P(i)$ in time polynomially bounded by $\log i$ and size(y). The optimization problem is *polynomial time solvable* for $P(i)$ for $i \in \mathbb{N}$ defined similarly, after replacing *separation* for *optimization.*

Theorem 2.9 *[45] For any class $P(i)$ where $i \in \mathbb{N}$, the separation problem is polynomially time solvable if and only if the optimization problem is polynomial time solvable.*

Clearly, one can solve the separation problem by checking each constraint but for problems where the number of constraints is exponential in size such a method is not polynomial time. For problems we consider in this thesis, efficient separation oracles have been obtained and we will give details whenever such an occasion arises.

2.3 Definitions

Laminar Family. Given a set V, $\mathcal{L} \subseteq 2^V$, a collection of subsets of V, is called *laminar* if for each $A, B \in \mathcal{L}$ we have either $A \cap B = \phi$ or $A \subseteq B$ or $B \subseteq A$. The following proposition about the size of a laminar family are standard and will be used in later chapters.

Proposition 2.10 *A laminar family $\mathcal{L}$ over the ground set V without singletons (subsets with only one element) has at most $|V| - 1$ distinct members.*

Proof: The proof is by induction on the size of the ground set. If $|V| = 2$, clearly the claim follows. Let $n = |V|$ and the claim be true for all laminar families over ground sets of size strictly smaller than n. Let S be a maximal set in the laminar family which is not equal to V. Each set in $\mathcal{L}$, except for V, is either contained in S or does not intersect S. The number of sets in $\mathcal{L}$ contained in S (including S itself) is at most $|S| - 1$ by the induction hypothesis. The sets in $\mathcal{L}$ not intersecting with S form a laminar family over the ground set $V \setminus S$ and hence there are at most $|V| - |S| - 1$ such sets. Along with V, this gives a total of at most $|S| - 1 + |V| - |S| - 1 + 1 = |V| - 1$ sets. □

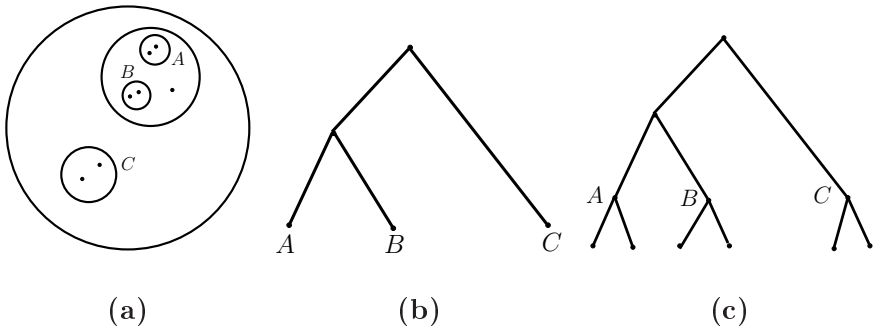


Figure 2.1: In Figure (a), the laminar family $\mathcal{L}$ over base set U. In Figure (b), the forest corresponding to $\mathcal{L}$ when singletons are not part of $\mathcal{L}$. Here $|\mathcal{L}| = 5$ and $|U| = 7$. In Figure (c), the forest corresponding to $\mathcal{L}$ when singletons are part of $\mathcal{L}$. Here $|\mathcal{L}| = 12$ and $|U| = 7$.

The following corollary follows immediately from Proposition 2.10 since addition of singleton sets to the family can add at most $|V|$ new members.

Corollary 2.11 *A laminar family $\mathcal{L}$ over the ground set V has at most $2|V| - 1$ distinct members.*

The laminar family $\mathcal{L}$ defines a directed forest $F_{\mathcal{L}}$ in which nodes correspond to sets in $\mathcal{L}$ and there exists an edge from set R to set S if R is the smallest set containing S. We call R the *parent* of S and S the *child* of R. A parent-less node is called a *root* and a childless node is called a *leaf*. Given a node R, the *subtree rooted at R* consists of R and all its descendants. We will abuse notation and use $\mathcal{L}$ to represent both the set family and the forest $F_{\mathcal{L}}$.

Cross-Free Family. A pair of sets $A, B \subseteq V$ are *crossing* if all of the sets $A \cap B, A - B, B - A, V - (A \cup B)$ are nonempty, and a family of sets $\mathcal{L} = \{A_1, A_2, \ldots, A_\ell\}$ is *cross-free* if no two of its sets are crossing.

Chain. A family $\mathcal{L} \subseteq 2^S$ is a chain if $A \in \mathcal{L}, B \in \mathcal{L}$, then $A \subseteq B$ or $B \subseteq A$. Observe that every chain is a laminar family but not vice-versa.

For subsets $A, B \subseteq U$ we define $A \Delta B = (A \setminus B) \cup (B \setminus A)$.

Graph Notation. Given a undirected graph $G = (V, E)$ and a subset of vertices $S \subseteq V$ we let $\delta(S)$ denote the set of edges with exactly one endpoint in S, i.e., $\delta(S) = \{e \in E : |e \cap S| = 1\}$. We also denote $\delta(\{v\})$ by $\delta(v)$ for a vertex $v \in V$. For subsets of vertices $S, T \subseteq V$, let $E(S, T)$ denote the set of edges with one endpoint in S and other in T and

$E(S) = E(S,S)$ be the set of edges with both endpoints in S. For an edge $e \in E$, the graph $G \setminus e$ denotes the graph obtained after deleting the edge e and the graph G/e denotes the graph obtained after contracting the endpoints of e (and deleting e). The graph $G \setminus v$ denotes the graph obtained after deleting the vertex v from G and each edge incident at v. For a set $F \subseteq E$, let $\chi(F)$ denote the vector in $\mathbb{R}^{|E|}$: the vector has an 1 corresponding to each edge $e \in F$, and 0 otherwise. This vector is called the *characteristic vector* of F, and is denoted by $\chi(F)$.

Given a directed graph $D = (V, A)$ and a subset of vertices $S \subseteq V$ we let $\delta^{in}(S)$ denote the set of arcs with their head in S and tail in S^c, i.e. $\delta^{in}(S) = \{(u,v) \in A : u \in S, v \notin S\}$. We let $\delta^{out}(S) = \{(u,v) \in A : u \notin S, v \in S\}$. For subsets $S, T \subseteq V$ we let $E(S,T) = \{(u,v) \in A : u \in S, v \in T\}$. The graph $D \setminus a$ denotes the directed graph formed after deleting arc a and graph $D/(u,v)$ is the graph formed after contracting the arc (u,v) in a single vertex.

Function Notation. A function $f : 2^V \to \mathbb{R}$ is called *submodular* if for each subset $S, T \subseteq V$ we have

$$f(S) + f(T) \geq f(S \cup T) + f(S \cap T)$$

while it is *supermodular* if for each subset $S, T \subseteq V$ we have

$$f(S) + f(T) \leq f(S \cup T) + f(S \cap T).$$

3

Exact Linear Programming Formulations

In this chapter we study various problems where an explicit linear description of all feasible integer solutions for the problem is known. In other words, there are explicit linear programming relaxations for these problems which are integral. These linear programming relaxations are well-studied in literature and have been shown to be integral via different proof techniques. We will use the iterative method to give yet another proof of integrality of these relaxations. In later chapters of the thesis, these iterative proofs will act as a stepping stone for achieving approximation algorithms for constrained versions of problems considered here.

All proofs follow the same overall outline we describe now. First we give the natural linear programming relaxation for the problem which is known to be integral. The second step is to give a characterization of any vertex solution through independent tight constraints which define the vertex uniquely. This step involves using the uncrossing technique. Although these characterization results are standard, we prove them here for completeness and to illustrate the uncrossing method. The most important ingredient of the proof is the final step, where we give a simple iterative procedure which constructs an integral solution using the optimum solution to the linear program. The iterative procedure, in each step, either selects an element in the integral solution which the linear program sets to a value of 1 or, deletes an element which the linear program sets to a value of 0. The technical heart of the argument involves showing that the procedure can always find such an element which the linear program sets to an integral value (0 or 1). This involves crucially using the characterization of the vertex solution by tight independent constraints.

We consider the bipartite matching problem in Section 3.1, the minimum spanning tree problem in Section 3.2, the minimum arborescence problem in Section 3.3, the minimum matroid basis problem in Section 3.4 and the perfect matching problem in Section 3.5.

3.1 Matching in Bipartite Graphs

Given a graph $G = (V, E)$, a subgraph H of G is called a *matching* if $d_H(v) \leq 1$ for all $v \in V$. The matching is called *perfect* if $d_H(v) = 1$ for all $v \in V$. Given a weight function $w : E \rightarrow \mathbb{R}$ the weight of a matching is defined to be the sum of the weights of the edges in the matching. A maximum weight perfect matching is a perfect matching of maximum weight.

Finding matchings in bipartite graphs is a fundamental problem which has played a crucial role in development of combinatorial optimization. Kuhn [67] building on the work of Egerváry [32] gave the famous Hungarian method which gives a polynomial time algorithm for the problem. Birkhoff [8] noted that the linear program for the bipartite matching problem given in Figure 3.1 is integral. Other proofs to show the integrality of the linear program have been given by von Neumann [102, 103], Dantzig [22], Hoffman and Wielandt [50], Koopmans and Beckmann [64], Hammersley and Mauldon [46], Tompkins [98] and Mirsky [76] (see the survey by Mirsky [77] and Schrijver [91], Chapter 18 and references therein). In this section, we use the iterative method to give a new proof that the linear programming relaxation for the bipartite matching problem is integral. This proof will serve as a stepping stone for obtaining an approximation algorithm for the Generalized Assignment problem in Section 7.1.

3.1.1 Linear Program

Given a bipartite graph $G = (V_1 \cup V_2, E)$ and a weight function $w : E \rightarrow \mathbb{R}$, the linear programming relaxation for the maximum weight bipartite matching problem is given by the following $LP_{BM}(G)$.

$$\begin{array}{lrcll} \text{maximize} & w(x) & = & \displaystyle\sum_{e \in E} w_e \, x_e & \\ \text{subject to} & \displaystyle\sum_{e \in \delta(v)} x_e & \leq & 1 & \forall \, v \in V_1 \cup V_2 \\ & x_e & \geq & 0 & \forall \, e \in E \end{array}$$

Figure 3.1: The Bipartite Matching Linear Program.

Polynomial Time Solvability. Observe that the linear program $LP_{BM}(G)$ is compact, i.e., the number of constraints and variables is polynomially bounded in the size

of the problem. Hence, the linear program can be solved optimally in polynomial time using the Ellipsoid algorithm or interior point algorithms (see Section 2.2.2).

We prove the following theorem by an iterative algorithm in the next section.

Theorem 3.1 *Given any weight function w there exists an integral matching M such that $w(M) \geq w \cdot x$ where x is the optimal solution to $LP_{BM}(G)$.*

Observe that the Theorem 3.1 as a corollary implies the following theorem.

Theorem 3.2 *The linear programming relaxation $LP_{BM}(G)$ is integral.*

3.1.2 Characterization of Vertex Solutions

Before we prove Theorem 3.1 we give a characterization of vertex solutions of LP_{BM} in the following lemma which follows by a direct application of the Rank Lemma.

Lemma 3.3 *Given any vertex solution x of linear program $LP_{BM}(G)$ such that $x_e > 0$ for each $e \in E$ there exists $W \subseteq V_1 \cup V_2$ such that*

1. *$x(\delta(v)) = 1$ for each $v \in W$ and x is the unique solution to the constraints $\{x(\delta(v)) = 1 : v \in W\}$.*
2. *The vectors $\{\chi(\delta(v)) : v \in W\}$ are linearly independent.*
3. *$|W| = |E|$.*

3.1.3 Iterative Algorithm

We now give the algorithm which constructs an integral matching of weight at least the optimal solution to $LP_{BM}(G)$ proving Theorem 3.1. The algorithm is a simple iterative procedure and shown in Figure 3.2.

We prove the correctness of the algorithm in two steps. First, we show that the algorithm returns a matching of optimal weight if the algorithm *always* finds an edge e with $x_e = 0$ in Step 2a or an edge e with $x_e = 1$ in Step 2b . In the second part, we show that the algorithm will always find such an edge completing the proof.

Iterative Bipartite Matching Algorithm

1. Initialization $F \leftarrow \emptyset$.
2. While $E(G) \neq \emptyset$
   (a) Find a vertex optimal solution x of $LP_{BM}(G)$ and remove every edge e with $x_e = 0$ from G.
   (b) If there is an edge $e = \{u, v\}$ such that $x_e = 1$ then update $F \leftarrow F \cup \{e\}$, $G \leftarrow G \setminus \{u, v\}$.
3. Return F.

Figure 3.2: Bipartite Matching Algorithm.

Claim 3.4 *If the algorithm, in every iteration, finds an edge e with $x_e = 0$ in Step 2a or an edge e with $x_e = 1$ in Step 2b, then it returns a matching F of weight at least the optimal solution to $LP_{BM}(G)$.*

Proof: The proof will proceed by induction on the number of iterations of the algorithm. The base case is trivial when the algorithm proceeds for only one iteration.

If we find an edge e such that $x_e = 0$ in Step 2a of the algorithm, then the residual problem is to find a matching in the graph $G' = G \setminus \{e\}$. The residual solution x_{res}, x restricted to G', is a feasible solution to the linear programming relaxation of the residual problem. By induction, the algorithm returns a matching $F' \subseteq E(G')$ with weight at least the optimal solution to $LP_{BM}(G')$. Since $w(F') \geq w \cdot x_{res} = w \cdot x$, the induction hypothesis holds in this case.

In the other case, if we find an edge $e = \{u, v\}$ such that $x_e = 1$ in Step 2b of the algorithm then the residual problem is to find a matching which contains the edge e. This is exactly the matching problem in graph $G' = G \setminus \{u, v\}$. Moreover x_{res}, x restricted to edges in G', is a feasible solution to the linear programming relaxation for the residual problem. Inductively, the algorithm will return a matching F' of weight at least the weight of the optimum solution of $LP_{BM}(G')$, and hence $w(F') \geq w \cdot x_{res}$, since x_{res} is a feasible solution to $LP_{BM}(G')$. The algorithm returns the matching $F = F' \cup \{e\}$ and we have

$$w(F) = w(F') + w_e \text{ and } w(F') \geq w \cdot x_{res}$$

which implies that

$$w(F) \geq w \cdot x_{res} + w_e = w \cdot x$$

since $x_e = 1$. Therefore, the weight of the matching returned by the algorithm is at least the weight of the LP solution x, which is a lower bound on the optimal weight. □

We now complete the proof of Theorem 3.1 by showing that the algorithm always finds an edge e such that $x_e = 0$ or $x_e = 1$. The proof of the following lemma crucially uses the characterization of vertex solutions given in Lemma 3.3.

Lemma 3.5 *Given any vertex solution x of $LP_{BM}(G)$ there must exist an edge e such that $x_e = 0$ or $x_e = 1$.*

Proof: Suppose for sake of contradiction $0 < x_e < 1$ for each edge $e \in E$. Lemma 3.3 implies that there exists $W \subseteq V_1 \cup V_2$ such that constraints corresponding to W are linearly independent and tight and $|E| = |W|$.

Claim 3.6 *We must have $deg_E(v) = 2$ for each $v \in W$ and $deg_E(v) = 0$ for each $v \notin W$.*

Proof: Firstly, $deg_E(v) \geq 2$ for each $v \in W$ else we have $x_e = 1$ or $x_e = 0$ for some edge $e \in E$ since $x(\delta(v)) = 1$ for each $v \in W$. But then we have

$$2|W| = 2|E| = \sum_{v \in V} deg_E(v) = \sum_{v \in W} deg_E(v) + \sum_{v \notin W} deg_E(v) \quad (3.1)$$

$$\geq 2|W| + \sum_{v \notin W} deg_E(v) \quad (3.2)$$

which implies that equality must hold everywhere in $deg_E(v) \geq 2$ for each $v \in W$ and $deg_E(v) = 0$ for each $v \notin W$. □

Hence, E is a cycle cover of vertices in W. Let C be any such cycle with all vertices in W. Since C is an even cycle because G is bipartite we also have

$$\sum_{v \in C \cap V_1} \chi(\delta(v)) = \sum_{v \in C \cap V_2} \chi(\delta(v))$$

which contradicts the independence of constraints in condition (2) of Lemma 3.3. □

Thus we obtain from Lemma 3.5 that the Algorithm in Figure 3.2 returns a matching which weighs at least the weight of the linear program. This completes the proof of Theorem 3.1.

3.2 Minimum Spanning Trees

Given a graph $G = (V, E)$, a spanning tree is a minimal connected subgraph of G. In an instance of the minimum spanning tree problem, we are given a graph with a cost function $c : E \to \mathbb{R}$ on the edges and the goal is to find a spanning tree of minimum cost.

The minimum spanning tree problem is a fundamental optimization problem and is a poster child problem for illustrating greedy algorithms, many variants of which have been obtained, starting from Boruvka [10], Kruskal [66] and Prim [83]. Edmonds [29] gave a linear programming relaxation (see Figure 3.3) for the minimum spanning tree problem and showed that it is integral. Various other proofs have been given to show the integrality of the linear program . Indeed, the famous spanning tree algorithms of Boruvka, Prim and Kruskal [91] can all be interpreted as *primal-dual* algorithms and thus imply the integrality of the linear program. It follows from Edmonds [27, 29] that the linear description in Figure 3.3 is totally dual integral (TDI) and thus as a corollary imply that the linear program is integral. Algorithms for decomposing fractional solutions into convex combination of spanning trees [20, 82] also imply the integrality of the linear program. We refer the reader to Magnanti and Wolsey [74] for a survey on spanning trees.

In this section we will give another proof of the integrality of linear program in Figure 3.3 using the iterative method. Actually, we give two iterative algorithms which show that the linear program is integral. In Chapter 4, both these algorithms are extended to algorithms for the degree constrained spanning tree problem . The first iterative algorithm extends to the MINIMUM BOUNDED-DEGREE SPANNING TREE problem when only upper bounds on the degrees are present in Section 4.1. The second algorithm extends when both upper and lower degree bounds are present in Section 4.3.

3.2.1 Linear Program

A linear programming relaxation for the minimum spanning tree problem, given by Edmonds [29], is shown in Figure 3.3. The linear program is also related to the study of the Travelling Salesman Problem. Recall that for a set $S \subseteq V$, we denote $E(S)$ to be the set of edges with both endpoints in S. The linear program $LP_{ST}(G)$ is as follows.

Observe that in any spanning tree T and subset $S \subseteq V$ we have $E(T) \cap E(S) \leq |S| - 1$ and hence the above linear program is a relaxation to the minimum spanning tree problem. We will show the integrality of the linear program using the iterative method.

$$\text{minimize} \quad \sum_{e \in E} c_e \, x_e$$

$$\begin{array}{rcll} x(E(S)) & \leq & |S| - 1 & \forall \; S \subset V \\ x(E(V)) & = & |V| - 1 & \\ x_e & \geq & 0 & \forall \; e \in E \end{array}$$

Figure 3.3: Linear program for minimum spanning tree problem.

Theorem 3.7 *[29] Every vertex solution to $LP_{ST}(G)$ is integral and corresponds to the characteristic vector of a spanning tree.*

Before we give the iterative algorithm and proof of Theorem 3.7, we show that one can optimize over the linear program $LP_{ST}(G)$ in polynomial time. We show this by giving a polynomial time separation oracle for the constraints in the linear program $LP_{ST}(G)$ (see Magnanti and Wolsey [74]). Alternatively, a compact equivalent linear program was given by Wong [107] thus implying that the linear program $LP_{ST}(G)$ is solvable in polynomial time.

Theorem 3.8 *There is a polynomial time separation oracle for the constraints in the linear program $LP_{ST}(G)$.*

Proof: Given a fractional solution x the separation oracle needs to find a set $S \subseteq V$ such that $x(E(S)) > |S| - 1$ if such a set exists. It is easy to check the equality $x(E(V)) = |V| - 1$. Thus, checking the inequality for each subset $S \subset V$ is equivalent to checking $\min_{S \subset V}\{|S| - 1 - x(E(S))\} < 0$. Using $x(E(V)) = |V| - 1$ we obtain that it is enough to check $\min_S\{|S| - 1 + x(E(V)) - x(E(S))\} < |V| - 1\}$. We show that solving $2|V| - 2$ min-cut problems suffice to check the above.

Fix a root vertex $r \in V$. For each $k \in V \setminus \{r\}$, we construct two minimum cut instances, one which checks the inequality for all subsets S containing r but not k and the other checks the inequality for all subsets S containing k but not r. We outline the construction for the first one, the second construction follows by changing the roles of r and k.

We construct a directed graph $\hat{G}$ with vertex set V and arcs (i, j) and (j, i) for each edge $\{i, j\}$ in G. We let the weight of edges (i, j) and (j, i) to be $\frac{x_{\{i,j\}}}{2}$. We also place arcs from each vertex $v \in V \setminus \{r, k\}$ to k of weight 1 and arcs from r to each vertex $v \in V \setminus \{r\}$

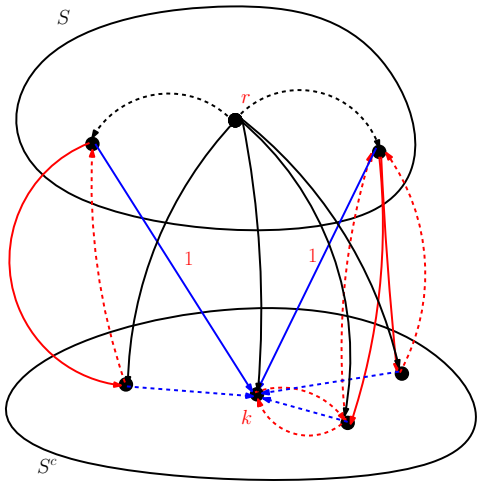


Figure 3.4: Blue edges into k have cost 1 and red edges between i and j cost $\frac{x_{\{i,j\}}}{2}$. Black edges from r to j cost $\sum_{e\in\delta(j)} \frac{x_e}{2}$. The solid edges are part of the cut and the dotted edges are not.

of weight $\sum_{e\in\delta(v)} \frac{x_e}{2} = \frac{x(\delta(v))}{2}$. Consider any cut $(S, V \setminus S)$ which separates r from k. Edges of weight one contribute exactly $|S| - 1$. The edges between i and j of weight $\frac{x_{ij}}{2}$ contribute exactly $\frac{x(\delta(S))}{2}$. The edges from r to rest of the vertices contribute $\sum_{v\notin S} \frac{x(\delta(v))}{2}$. Thus the total weight of the cut is exactly

$$|S| - 1 + \frac{x(\delta(S))}{2} + \sum_{v\notin S} \frac{x(\delta(v))}{2} = |S| - 1 + x(E(V)) - x(E(S)).$$

Hence, checking whether the minimum cut separating r from k is strictly smaller than $|V| - 1$ checks exactly whether there is a violating set S not containing k but containing r. □

3.2.2 Characterization of Vertex Solutions

In this section we give a characterization of a vertex solution to the linear program $LP_{ST}(G)$ by a small set of tight independent constraints. There are exponentially many inequalities in the linear program $LP_{ST}(G)$, and a vertex solution may satisfy many inequalities as equalities. To analyze a vertex solution, an important step is to find a *small* set of tight inequalities defining it. If there is an edge e with $x_e = 0$, this edge can be removed from the graph without affecting the feasibility and the objective value. So henceforth assume every edge e has $x_e > 0$. The characterization is well-known (see Cornuejols et al [19],

Goemans [42]) and we include it here for completeness and to illustrate the uncrossing technique which will occur at multiple occasions in the thesis.

Uncrossing Technique

We use the uncrossing technique to find a *good* set of tight inequalities for a vertex solution in the linear program $LP_{ST}(G)$. As mentioned earlier, the uncrossing method is used extensively in combinatorial optimization and the discussion here will give an illustration of the method. Recall that $E(X,Y)$ denotes the set of edges with one endpoint in X and other in Y. The following proposition is straightforward and states the supermodularity of function $E(X)$.

Proposition 3.9 *For $X, Y \subseteq V$,*

$$\chi(E(X)) + \chi(E(Y)) \leq \chi(E(X \cup Y)) + \chi(E(X \cap Y)),$$

and equality holds if and only if $E(X \setminus Y, Y \setminus X) = \emptyset$.

Proof: Observe that

$$\chi(E(X)) + \chi(E(Y)) = \chi(E(X \cup Y)) + \chi(E(X \cap Y)) - \chi(E(X \setminus Y, Y \setminus X))$$

and proof follows immediately. (See Figure 3.5). □

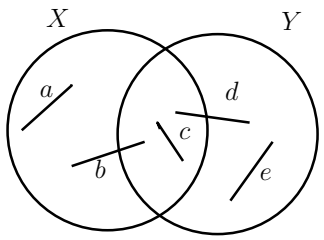


Figure 3.5: Edges labelled a, b, d and e are counted once both in LHS and RHS while edges labelled c are counted twice.

Given a vertex solution x of the linear program $LP_{ST}(G)$, let $\mathcal{F} = \{S \mid x(E(S)) = |S| - 1\}$ be the family of tight inequalities for a vertex solution x in the linear program $LP_{ST}(G)$. The following lemma shows that the family $\mathcal{F}$ can be *uncrossed*.

Lemma 3.10 *If $S, T \in \mathcal{F}$ and $S \cap T \neq \emptyset$, then both $S \cap T$ and $S \cup T$ are in $\mathcal{F}$. Furthermore, $\chi(E(S)) + \chi(E(T)) = \chi(E(S \cap T)) + \chi(E(S \cup T))$.*

Proof: Observe that

$$\begin{aligned} |S| - 1 + |T| - 1 &= x(E(S)) + x(E(T)) \\ &\leq x(E(S \cap T)) + x(E(S \cup T))) \\ &\leq |S \cap T| - 1 + |S \cup T| - 1 \\ &= |S| - 1 + |T| - 1. \end{aligned}$$

The first equality follows from the fact that $S, T \in \mathcal{F}$. The second inequality follows from Proposition 3.9. The third inequality follows from constraints for $S \cap T$ and $S \cup T$ in the linear program $LP_{ST}(G)$. The last equality is because $|S| + |T| = |S \cap T| + |S \cup T|$ for any two sets S, T.

Equality must hold everywhere above and we have $x(E(S \cap T)) + x(E(S \cup T)) = |S \cap T| - 1 + |S \cup T| - 1$. Thus, we must have equality for constraints for $S \cap T$ and $S \cup T$, i.e., $x(E(S \cap T)) = |S \cap T| - 1$ and $x(E(S \cup T)) = |S \cup T| - 1$, which implies that $S \cap T$ and $S \cup T$ are also in $\mathcal{F}$. Moreover, equality holds for Proposition 3.9 and thus $\chi(E(S \setminus T, T \setminus S)) = \emptyset$ and $\chi(E(S)) + \chi(E(T)) = \chi(E(S \cap T)) + \chi(E(S \cup T))$. □

Denote by $span(\mathcal{F})$ the vector space generated by the set of vectors $\{\chi(E(S)) \mid S \in \mathcal{F}\}$. The following lemma states that a maximal set of independent tight constraints can be chosen to form a laminar family.

Lemma 3.11 *[53] If $\mathcal{L}$ is a maximal laminar subfamily of $\mathcal{F}$, then $span(\mathcal{L}) = span(\mathcal{F})$.*

Proof: Suppose, by way of contradiction, that $\mathcal{L}$ is a maximal laminar subfamily of $\mathcal{F}$ but $span(\mathcal{L}) \subsetneq span(\mathcal{F})$. For any $S \notin \mathcal{L}$, define $intersect(S, \mathcal{L})$ to be the number of sets in $\mathcal{L}$ which intersect S, i.e. $intersect(S, \mathcal{L}) = |\{T \in \mathcal{L} \mid S \text{ and } T \text{ intersect}\}|$. Since $span(\mathcal{L}) \subset span(\mathcal{F})$, there exists a set $S \in \mathcal{F}$ with $\chi(E(S)) \notin span(\mathcal{L})$. Choose such a set S with minimum $intersect(S, \mathcal{L})$. Clearly, $intersect(S, \mathcal{L}) \geq 1$; otherwise $\mathcal{L} \cup \{S\}$ is also a laminar subfamily, contradicting the maximality of $\mathcal{L}$. Let T be a set in $\mathcal{L}$ which intersects S. Since $S, T \in \mathcal{F}$, by Lemma 3.10, both $S \cap T$ and $S \cup T$ are in $\mathcal{F}$. Also, both $intersect(S \cap T, \mathcal{L})$ and $intersect(S \cup T, \mathcal{L})$ are smaller than $intersect(S, \mathcal{L})$, proved in Proposition 3.12. Hence, by the minimality of $intersect(S, \mathcal{L})$, both $S \cap T$ and $S \cup T$ are in $span(\mathcal{L})$. By Lemma 3.10, $\chi(E(S)) + \chi(E(T)) = \chi(E(S \cap T)) + \chi(E(S \cup T))$. Since

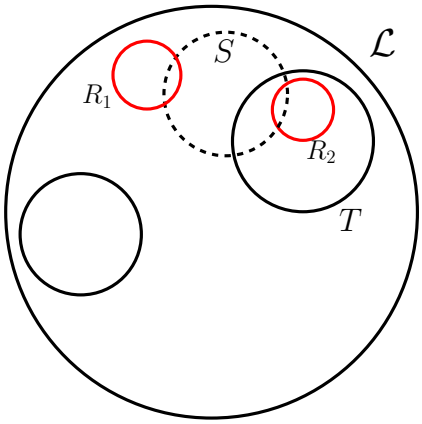


Figure 3.6: The figure illustrates the different cases for sets R, R_1 and R_2, which intersect S non-trivially and are in the laminar family $\mathcal{L}$.

$\chi(E(S \cap T)), \chi(E(S \cup T))$ are in $span(\mathcal{L})$ and $T \in \mathcal{L}$, the above equation implies that $\chi(E(S)) \in span(\mathcal{L})$, a contradiction. It remains to prove Proposition 3.12.

Proposition 3.12 *Let S be a set that intersects $T \in \mathcal{L}$. Then $intersect(S \cap T, \mathcal{L})$ and $intersect(S \cup T, \mathcal{L})$ are smaller than $intersect(S, \mathcal{L})$.*

Proof: Since $\mathcal{L}$ is a laminar family, for a set $R \in \mathcal{L}$ with $R \neq T$, R does not intersect T (either $R \subset T$, $T \subset R$ or $T \cap R = \emptyset$). So, whenever R intersects $S \cap T$ or $S \cup T$, R also intersects S. See Figure 3.6 for different cases of R. Also, T intersects S but not $S \cap T$ or $S \cup T$. Therefore, $intersect(S \cap T, \mathcal{L})$ and $intersect(S \cup T, \mathcal{L})$ are smaller than $intersect(S, \mathcal{L})$ by at at least one (i.e. T). □

This completes the proof of Lemma 3.11. □

Thus we obtain the following characterization using the Rank Lemma and the Lemma 3.11.

Lemma 3.13 *Let x be a vertex solution of the linear program $LP_{ST}(G)$ such that $x_e > 0$ for each edge e and let $\mathcal{F} = \{S \subseteq V : x(E(S)) = |S| - 1\}$ be the set of all tight constraints. Then there exists a laminar family $\mathcal{L} \subseteq \mathcal{F}$ such that*

1. *The vectors $\{\chi(E(S)) : S \in \mathcal{L}\}$ are linearly independent.*
2. *$span(\mathcal{L})$=$span(\mathcal{F})$.*

3. $|\mathcal{L}| = |E|$.

3.2.3 Iterative Algorithm

In this subsection, an iterative procedure to find a minimum spanning tree from a vertex optimal solution of the linear program $LP_{ST}(G)$ is presented. The procedure is generalized in Section 4.1 when degree constraints are present. The algorithm is shown in Figure 3.7.

Iterative MST Algorithm

1. Initialization $F \leftarrow \emptyset$.
2. While $V(G) \neq \emptyset$
   (a) Find a vertex optimal solution x of the linear program $LP_{ST}(G)$ and remove every edge e with $x_e = 0$ from G.
   (b) Find a vertex v with exactly one edge $e = uv$ incident on it, and update $F \leftarrow F \cup \{e\}$, $G \leftarrow G \setminus \{v\}$.
3. Return F.

Figure 3.7: Iterative MST Algorithm.

We first show that the algorithm always finds a *leaf* vertex in Step 2b. Then we show that the solution returned by the algorithm is optimal.

Lemma 3.14 *For any vertex solution x of the linear program $LP_{ST}(G)$ with $x_e > 0$ for every edge e, there exists a vertex v with $deg(v) = 1$.*

Proof: Suppose each vertex is of degree at least two in the support E. Then $|E| = \frac{1}{2}\sum_{v \in V} deg(v) \geq |V|$. Since there is no edge e with $x_e = 0$, every tight inequality is of the form $x(E(S)) = |S| - 1$. Recall from Lemma 3.13 there there is a laminar family $\mathcal{L}$ with $|E| = |\mathcal{L}|$. By Proposition 2.10, $|\mathcal{L}| \leq |V| - 1$ and hence $|E| \leq |V| - 1$, a contradiction. □

The remaining thing to check is the returned solution is a minimum spanning tree, which is proved in the following theorem.

Theorem 3.15 *The Iterative MST Algorithm returns a minimum spanning tree in polynomial time.*

Proof: This is proved by induction on the number of iterations of the algorithm. If the algorithm finds a vertex v of degree one (a leaf vertex) in Step 2(b) with an edge $e = \{u, v\}$ incident at v, then we must have $x_e = 1$ since $x(\delta(v)) \geq 1$ is a valid inequality for the linear program (Subtract the constraint $x(E(V - v)) \leq |V| - 2$ from the constraint $x(E(V)) = |V| - 1$). Intuitively, v is a leaf of the spanning tree. Thus, e is added to the solution F (initially $F = \emptyset$), and v is removed from the graph. Note that for any spanning tree T' of $G' = G \setminus \{v\}$, a spanning tree $T = T' \cup \{e\}$ of G can be constructed. Hence, the residual problem is to find a minimum spanning tree on $G \setminus v$, and the same procedure is applied to solve the residual problem recursively. Observe that the restriction of x to $E(G')$, denoted by x_{res}, is a feasible solution to the linear program $LP_{ST}(G')$. By induction, the algorithm will return a spanning tree F' of G' of cost at most the optimal value of the linear program $LP_{ST}(G')$, and hence $c(F') \leq c \cdot x_{res}$, as x_{res} is a feasible solution to the linear program $LP_{ST}(G')$. Therefore,

$$c(F) = c(F') + c_e \text{ and } c(F') \leq c \cdot x_{res}$$

which imply that

$$c(F) \leq c \cdot x_{res} + c_e = c \cdot x$$

as $x_e = 1$. Hence, the spanning tree returned by the algorithm is of cost no more than the cost of an optimal LP solution x, which is a lower bound on the cost of a minimum spanning tree. This shows that the algorithm returns a minimum spanning tree of the graph. □

Remark. If x is an optimal vertex solution to the linear program $LP_{ST}(G)$ for G, then the residual LP solution x_{res}, x restricted to $G' = G \setminus v$, remains an optimal vertex solution to the linear program $LP_{ST}(G')$. Hence, in the Iterative MST Algorithm we only need to solve the original linear program once and none of the residual linear programs.

Theorem 3.15 also shows that the linear program $LP_{ST}(G)$ is an exact formulation of the minimum spanning tree problem showing the proof of Theorem 3.7.

3.2.4 Iterative Algorithm II

In this section, we give another iterative procedure to find a minimum spanning tree from a vertex optimal solution of the linear program $LP_{ST}(G)$ is presented. The alternate method is useful is addressing degree constraints in Section 4.3. The algorithm is shown in Figure 3.8.

Iterative MST Algorithm II

1. Initialization $F \leftarrow \emptyset$.
2. While $V(G) \neq \emptyset$
   (a) Find a vertex optimal solution x of the linear program $LP_{ST}(G)$ and remove every edge e with $x_e = 0$ from G.
   (b) Find an edge $e = \{u, v\}$ such that $x_e = 1$ and update $F \leftarrow F \cup \{e\}$, $G \leftarrow G/e$.
3. Return F.

Figure 3.8: Iterative MST Algorithm II.

Following the discussion in Section 3.2.3 it is enough to show that the algorithm will terminate. An argument similar to one in proof of Theorem 3.15 will show that the output of the algorithm is a minimum spanning tree.

Lemma 3.16 *For any vertex solution x of the linear program $LP_{ST}(G)$ with $x_e \geq 0$ for each edge e there exists an edge f such that $x_f = 1$.*

Proof: Lemma 3.14 already gives one proof of the fact by showing that there exists a vertex v such that $deg(v) = 1$. We give two other alternate proofs of this lemma.

Proof 1. Lemma 3.13 and Proposition 2.10 shows that $|E| = |V| - 1$ and since $x(E) = |V| - 1$ and $x(e) \leq 1$ for all edges $e \in E$ (by considering the constraint $x(E(S)) = |S| - 1$ for a size two set S), we must have $x_e = 1$ for all edges $e \in E$ proving integrality. Thus we have that directly either $x_e = 0$ or $x_e = 1$ for all edges e rather than for a single edge.

Proof 2. Observe that by Lemma 3.13, there are $|\mathcal{L}|$ linearly independent tight constraints of the form $x(E(S)) = |S| - 1$ with $|E| = |\mathcal{L}|$. We now show a contradiction to this through a counting argument.

We assign one token for each edge e to the smallest set containing both the endpoints. Thus, we assign a total of $|E|$ tokens. Now, we collect at least one token for each set in $\mathcal{L}$ and some extra tokens giving us a contradiction to Lemma 3.13. Actually, we collect two tokens for each set $S \in \mathcal{L}$. Let S be any set in $\mathcal{L}$ with children $R_1, \ldots, R_k$ where $k \geq 0$. We have

$$x(E(S)) = |S| - 1$$

and for each i,

$$x(E(R_i)) = |R_i| - 1$$

Subtracting, we obtain

$$x(E(S)) - \sum_i x(E(R_i)) = |S| - \sum_i |R_i| + k - 1$$
$$\implies x(A) = |S| - \sum_i |R_i| + k - 1$$

where $A = E(S) \setminus (\cup_i E(R_i))$. Observe that S obtains exactly one token for each edge in A. If $A = \emptyset$, then $\chi(E(S)) = \sum_i \chi(E(R_i))$ which contradicts the independence of constraints. Moreover, $|A| \neq 1$ as $x(A)$ is an integer and each x_e is fractional. Hence, S receives at least two tokens. □

This completes the proof of Lemma 3.16 showing the correctness of Iterative Algorithm II.

3.3 Arborescences

Given a directed graph D and a root vertex r, a (spanning) r-arborescence is a subgraph of D so that there is a directed path from r to every vertex in $V - r$. The *minimum cost (spanning) arborescence* problem is to find a spanning r-arborescence with minimum total cost. The problem generalizes the minimum spanning tree problem and efficient algorithms for computing a minimum cost arborescence were given by Chu and Liu [15], Edmonds [27] and Bock [9]. Edmonds [28] gave an integral linear program for the minimum arborescence problem which we state in Figure 3.9. Integrality of the linear program follows directly from the algorithms for the minimum arborescence problem [9, 15, 27]. Edmonds and Giles [31] and Frank [35] used the uncrossing method on the dual of the linear program to show that the linear description in Figure 3.9 is totally dual integral.

In this section we give an iterative algorithm to show that the linear program in Figure 3.9 is integral. The iterative algorithm is then extended in Section 6.2 to give an approximation algorithm for the degree constrained arborescence problem.

3.3.1 Linear Program

We now give a linear program for the minimum arborescence problem given by Edmonds [28] in Figure 3.9. The linear program $LP_{arb}(D)$ requires that there is a directed path from a fixed vertex r to every vertex in $V - r$. Or equivalently, by Menger's theorem [75], it specifies that there is at least one arc entering every set which does not contain the root vertex.

$$\text{minimize} \quad \sum_{a \in A} c_a \, x_a$$

$$\begin{aligned} x(\delta^{in}(S)) &\geq 1 && \forall \; S \subseteq V - r \\ x(\delta^{in}(v)) &= 1 && \forall \; v \in V \setminus \{r\} \\ x(\delta^{in}(r)) &= 0 && \\ x_a &\geq 0 && \forall \; a \in A \end{aligned}$$

Figure 3.9: Linear Program for the Minimum Cost Arborescence.

Separation Oracle

Although the number of constraints is exponential in the size of the problem, the availability of an efficient separation oracle ensures the polynomial solvability of the linear progrma $LP_{arb}(D)$. The separation oracle is quite straightforward. Given any solution x, the separation oracle first constructs a graph with arc weights as x_a. It then computes the *min-cut* from the root vertex r to every other vertex. If every *min-cut* is at least 1, it is easy to see that the solution is feasible. If there exists a *min-cut* of value less than 1, the violating constraint is precisely the set of vertices that this cut separates. The equality constraints can be checked one by one since there is only one constraint for each vertex for a total of $|V|$ constraints.

Compact Formulation

Wong [107] and Maculan [73] observed that the minimum arborescence problem can be formulated as a compact linear programming problem. We now give the compact linear program in Figure 3.10. The basic ideas is to use the equivalence of flows and cuts. This compact formulation also shows how to solve the equivalent linear program in Figure 3.9 in polynomial time.

$$\text{minimize} \quad \sum_{a \in A} c_a \, x_a$$

$$\begin{aligned}
\sum_{a \in \delta^{in}(v)} f_a^v &= 1 && \forall\, v \in V - r, \\
\sum_{a \in \delta^{in}(v)} f_a^v - \sum_{a \in \delta^{out}(v)} f_a^v &= 0 && \forall\, v \in V - r, \forall\, u \in V - r - v, \\
\sum_{a \in \delta^{in}(v)} f_a^v - \sum_{a \in \delta^{out}(v)} f_a^v &= -1 && \forall\, v \in V - r, \\
x_a &\geq f_a^v && \forall\, a \in A, \forall\, v \in V - r, \\
x(\delta^{in}(v)) &= 1 && \forall\, v \in V \setminus \{r\} \\
f_a^v, x_a &\geq 0 && \forall\, a \in A, \forall\, v \in V - r,
\end{aligned}$$

Figure 3.10: Compact Linear Program for the Minimum Cost Arborescence.

3.3.2 Characterization of Vertex Solutions

As in the case of the minimum spanning tree problem in Section 3.2, the uncrossing technique is used to find a small set of tight inequalities that defines a vertex solution of the linear program $LP_{arb}(D)$. This characterization follows from results of Edmonds and Giles [31] and Frank [35] and we include the proofs here for completeness.

Let $\mathcal{F} = \{S \mid x(\delta^{in}(S)) = 1\}$ be the family of tight inequalities for a vertex solution x in the directed LP. The following proposition states the standard fact that cuts in a graph are submodular.

Proposition 3.17 *For $X, Y \subseteq V$,*

$$\chi(\delta^{in}(X)) + \chi(\delta^{in}(Y)) \geq \chi(\delta^{in}(X \cup Y)) + \chi(\delta^{in}(X \cap Y)),$$

and equality holds if and only if $E(X \setminus Y, Y \setminus X) = \emptyset$ and $E(Y \setminus X, X \setminus Y) = \emptyset$.

Proof: Observe that

$$\chi(\delta^{in}(X)) + \chi(\delta^{in}(Y)) = \chi(\delta^{in}(X \cup Y)) + \chi(\delta^{in}(X \cap Y)) + \chi(E(X \setminus Y, Y \setminus X)) + \chi(E(Y \setminus X, X \setminus Y)),$$

and the proof follows immediately. □

The following lemma shows that the family $\mathcal{F}$ can be uncrossed.

Lemma 3.18 *If $S, T \in \mathcal{F}$ and $S \cap T \neq \emptyset$, then both $S \cap T$ and $S \cup T$ are in $\mathcal{F}$. Furthermore, $\chi(\delta^{in}(S)) + \chi(\delta^{in}(T)) = \chi(\delta^{in}(S \cap T)) + \chi(\delta^{in}(S \cup T))$.*

Proof:

$$\begin{aligned} 1+1 &= x(\delta^{in}(S)) + x(\delta^{in}(T)) \\ &\geq x(\delta^{in}(S \cap T)) + x(\delta^{in}(S \cup T)) \\ &\geq 1+1 \end{aligned}$$

The first equality follows from the fact that $S, T \in \mathcal{F}$. The second inequality follows from Proposition 3.17. The third inequality follows from the constraints for $S \cap T$ and $S \cup T$ in the linear program $LP_{arb}(D)$.

Equality must hold everywhere and we have $x(\delta^{in}(S \cap T)) + x(\delta^{in}(S \cup T)) = 2$. Thus, we must have equality for constraints for $S \cap T$ and $S \cup T$, i.e., $x(\delta^{in}(S \cap T)) = 1$ and $x(\delta^{in}(S \cup T)) = 1$, which implies that $S \cap T$ and $S \cup T$ are also in $\mathcal{F}$. Moreover, equality holds for Proposition 3.17 and thus $\chi(E(S \setminus T, T \setminus S)) \cup \chi(E(T \setminus S, S \setminus T)) = \emptyset$ and $\chi(\delta^{in}(S)) + \chi(\delta^{in}(T)) = \chi(\delta^{in}(S \cap T)) + \chi(\delta^{in}(S \cup T))$. □

Denote by $span(\mathcal{F})$ the vector space generated by the set of vectors $\{\chi(\delta^{in}(S)) \mid S \in \mathcal{F}\}$. The following lemma says that a vertex solution is characterized by tight inequalities whose corresponding sets form a laminar family.

Lemma 3.19 *If $\mathcal{L}$ is a maximal laminar subfamily of $\mathcal{F}$, then $span(\mathcal{L}) = span(\mathcal{F})$.*

Proof: The proof follows the same lines as in the case of undirected spanning trees and is omitted. □

Thus we get as a corollary our main characterization result.

Lemma 3.20 *Let x be any vertex solution of the linear program $LP_{arb}(D)$ such that $x_a > 0$ for each $a \in A$. Then there exists a laminar family $\mathcal{L}$ such that x is the unique solution to the following linear system.*

$$x(\delta^{in}(S)) = 1 \quad \forall S \in \mathcal{L}.$$

Moreover, the characteristic vectors $\{\chi(\delta^{in}(S)) \ : \ S \in \mathcal{L}\}$ are linearly independent and $|\mathcal{L}| = |A|$

3.3.3 Iterative Algorithm

We now present the iterative algorithm to obtain an integral optimal solution to the LP in Figure 3.9. The algorithm is similar to the Iterative MST Algorithm in Section 3.2. The main difference is that after we pick an arc $a = uv$ with $x_a = 1$, we contract $\{u, v\}$ into a single vertex.

Iterative Arborescence Algorithm

1. Initialization $F \leftarrow \emptyset$.
2. While $V(D) \neq \emptyset$
   (a) Find a vertex optimal solution x of the linear program LP_{arb} and remove every arc a with $x_a = 0$ from D.
   (b) Find an arc $a = uv$ with $x_a = 1$, and update $F \leftarrow F \cup \{a\}$, $D \leftarrow D/\{uv\}$.
3. Return F.

Figure 3.11: Iterative Arborescence Algorithm.

As in minimum spanning trees, assuming the algorithm terminates successfully, it is easy to show that the returned solution is a minimum spanning arborescence.

Theorem 3.21 *The Iterative Arborescence Algorithm returns a minimum cost arborescence in polynomial time.*

Proof: The proof follows the same argument as in for undirected spanning trees. □

The key step is to prove that the algorithm will terminate. For the iterative relaxation technique to converge, we would need at each stage, to find an arc a with either $x_a = 1$ or $x_a = 0$ which we show in the following lemma.

Lemma 3.22 *For any vertex solution x of the directed LP, either there is an arc with $x_a = 0$ or there is an arc with $x_a = 1$.*

Before we begin the proof, let us recall that there exists a laminar family $\mathcal{L}$ such that it represents a linearly independent set of tight constraints(Lemma 3.20). The proof, by contradiction, is based on a token argument (as in earlier proofs). The idea of the argument is to assume that there is no arc with $x_a = 0$ and $x_a = 1$, and then derive a contradiction

by showing that the number of constraints (that is, the number of sets in $\mathcal{L}$) is smaller than the number of non zero variables (that is, the number of arcs) - contradicting the Lemma 3.20.

Two different counting arguments will be presented for the contradiction. The first argument is quite straightforward but we include a more involved argument which is extended to degree constrained arborescence problem in Section 6.2.

Counting Argument 1

Suppose for sake of contradiction $0 < x_a < 1$ for all $a \in A$. But, we have $x(\delta^{in}(v)) \geq 1$ for each $v \in V \setminus \{r\}$. Hence, we must have $|\delta^{in}(v)| \geq 2$ for each $v \in V \setminus \{r\}$. Thus

$$|A| = \sum_{v \in V} |\delta^{in}(v)| \geq \sum_{v \in V \setminus \{r\}} 2 = 2|V| - 2$$

But from Lemma 3.20 we have the maximal linearly independent constraints form a laminar family over the ground set $V \setminus \{r\}$. From Corollary 2.11 we have that $|\mathcal{L}| \leq 2(|V|-1)-1 = 2|V|-3$. But this contradicts the Rank Lemma since $|A| \geq 2|V|-2 > |\mathcal{L}|$.

Counting Argument 2

We now give another counting argument which is more involved but is useful in extending to Minimum Bounded-Degree Arborescence problem in Section 6.2.

For each arc, one *token* is assigned to its head. So the total number of tokens assigned is exactly $|E|$. These tokens will be redistributed such that each subset $S \in \mathcal{L}$ is assigned one token, and there are still some excess tokens left. This will imply $|E| > |\mathcal{L}|$ and thus contradicts Lemma 3.19. The following lemma shows that such a redistribution is possible.

Lemma 3.23 *For any rooted subtree of the forest $\mathcal{L} \neq \emptyset$ with root S, the tokens assigned to vertices inside S can be distributed such that every node in the subtree gets at least one token and the root S gets at least two tokens.*

Proof: The proof is by induction on the height of the subtree. The base case is when S is a leaf. Since $x(\delta^{in}(S)) = 1$ and there is no arc with $x_e = 1$, there are at least two arcs in $\delta^{in}(S)$, and therefore S gets at least two tokens.

For the induction step, let S be the root and $R_1, \ldots, R_k$ be its children. By the induction hypothesis, each node in the subtree rooted at R_i gets at least one token and R_i gets at least two tokens. Since R_i only needs to keep one token, it can give one token to S. Suppose $k \geq 2$, then S can collect two tokens by taking one token from each of its children, as required. So suppose $k = 1$. If there is an arc e which enters S but not to R_1, then S can collect two tokens by taking one token from R_1 and one token from e. Suppose such an arc does not exist, then $\delta^{in}(S) \subseteq \delta^{in}(R)$. Since $x(\delta^{in}(S)) = x(\delta^{in}(R)) = 1$ and there is no arc with $x_e = 0$, this implies $\delta^{in}(S) = \delta^{in}(R)$. Hence $\chi(\delta^{in}(S)) = \chi(\delta^{in}(R))$, but this contradicts the linear independence of the characteristic vectors for sets in $\mathcal{L}$ (recall that $\mathcal{L}$ can be chosen to satisfy the properties in Corollary 3.20). Therefore, such an arc must exist, and S can collect two tokens, as required. This completes the proof of the induction step. □

Applying Lemma 3.23 to each root of the forest $\mathcal{L}$ we obtain that the number of tokens is at least $|\mathcal{L}| + 1$ which implies that $|E| > |\mathcal{L}|$, contradicting Corollary 3.20. This completes the proof of Lemma 3.22, and hence Theorem 3.21 follows.

3.4 Matroid Basis

In this section we consider the minimum cost matroid basis problem. We first define the matroid structure, which generalizes spanning trees and a host of other interesting objects. Then, we give the linear programming relaxation, first given by Edmonds [30], for the minimum cost matroid basis problem. We then give an iterative algorithm to show the integrality of the linear program. This iterative algorithm is then extended in Section 7.4 to give approximation algorithms for the degree constrained matroid basis problem.

Matroids were introduced by Whitney [105] and equivalent systems were considered by Nakasawa [79], Birkhoff and van der Waerden [104].

Definition 3.24 *A pair $\mathcal{M} = (S, \mathcal{I})$ is a matroid if for $A, B \subseteq S$,*

1. *$A \in \mathcal{I}$ and $B \subseteq A \implies B \in \mathcal{I}$.*
2. *$A, B \in \mathcal{I}$ and $|B| > |A| \implies \exists x \in B \setminus A$ such that $A \cup \{x\} \in \mathcal{I}$.*

S is called the *ground set* of the matroid M. A set $A \subseteq S$ is *independent* if $A \in \mathcal{I}$ else it is called *dependent*. An inclusionwise maximal set $A \in \mathcal{I}$ is called a *basis* of M. Observe that Property 2 implies that all bases have the same cardinality.

Examples of Matroids

1. **Graphic Matroid [8, 105].** Given a connected graph $G = (V, E)$, the graphic matroid of G is defined as $\mathcal{M}_G = (E, \mathcal{I}_G)$ where $\mathcal{I}_G = \{F \subseteq E | F \text{ is a forest } \}$.

2. **Uniform Matroid.** Given a set S and an integer $k \geq 0$ the uniform matroid of rank k is defined as $\mathcal{M}_S^k = (S, \mathcal{I}^k)$ where $\mathcal{I}^k = \{T \subseteq S : |T| \leq k\}$.

3. **Linear Matroid [97].** Let A be an $m \times n$ matrix and $S = \{1, \ldots, n\}$. For any $1 \leq i \leq n$, let A^i denote the i^{th}-column of A. The linear matroid over matrix A is defined as $\mathcal{M}_A = (S, \mathcal{I}_A)$ where $\mathcal{I}_A = \{T \subseteq S : A^i \text{ for } i \in T \text{ are linearly independent}\}$.

4. **Matroid Restriction.** Let $\mathcal{M} = (S, \mathcal{I})$ be a matroid and $T \subseteq S$. Then the matroid restriction of $\mathcal{M}$ to the set T is the matroid $\mathcal{M}_T = (T, \mathcal{I}_T)$ where $\mathcal{I}_T = \{R : R \in \mathcal{I}, R \subseteq T\}$.

It is straightforward to verify that the above examples satisfy the properties of matroids and we refer the reader to Lawler [71], Schrijver [91, 94] for historical and technical details on matroids.

Definition 3.25 ***(Rank function)*** *Given a matroid $\mathcal{M} = (S, \mathcal{I})$, the rank function $r_{\mathcal{M}} : 2^S \rightarrow \mathbb{Z}$ of the matroid $\mathcal{M}$ is defined as $r_{\mathcal{M}}(T) = \max_{U \subseteq T, U \in \mathcal{I}} |U|$.*

We will drop the subscript $\mathcal{M}$ from the rank function $r_{\mathcal{M}}$ when the matroid $\mathcal{M}$ is clear from the context. Observe that $A \in \mathcal{I}$ iff $r(A) = |A|$. We also use the following important property about the rank function of matroids. Here we include a proof for completeness.

Lemma 3.26 *[105] Let r be the rank function of matroid $\mathcal{M} = (S, \mathcal{I})$. Then r is a submodular function, i.e., for all $A, B \subseteq S$, we have $r(A) + r(B) \geq r(A \cap B) + r(A \cup B)$.*

Proof: Let $r(A \cap B) = k_1$, $r(A \cup B) = k_2$. This implies that $\exists V \subseteq A \cap B$ such that $r(V) = |V| = k_1$. Similarly, there exists $U \subseteq A \cup B$ such that $r(U) = |U| = k_2$. Moreover, since every independent set can be extended to a basis, we can assume that $V \subseteq U$ and by definition of V we must have $U \cap A \cap B = V$. Since, U is independent, we have $r(A) \geq |U \cap A|$ and $r(B) \geq |U \cap B|$. Now, we have

$$|U \cap A| + |U \cap B| = |U \cap (A \cup B)| + |U \cap (A \cap B)|$$

$$\implies r(A) + r(B) \geq r(A \cup B) + r(A \cap B)$$

since $|U| \cap (A \cup B)| = k_1$ and $|U \cap (A \cap B)| = |V| = k_2$. □

We now define two important operations on matroids and their effect on the rank function. For details, we refer the reader to Oxley [81].

Definition 3.27 ***(Deletion)*** *Given a matroid $\mathcal{M} = (S, \mathcal{I})$ and $x \in S$ we define $M \setminus x = (S \setminus \{x\}, \mathcal{I}')$ where $\mathcal{I}' = \{T \setminus \{x\} : T \in \mathcal{I}\}$ to be the matroid obtained by deleting x from M. The rank function of M x, r_1 is related to the rank function r of M by the formula $r_1(T) = r(T)$ for $T \subseteq S \setminus \{x\}$.*

Definition 3.28 ***(Contraction)*** *Given a matroid $\mathcal{M} = (S, \mathcal{I})$ and $x \in S$ we define $M/x = (S \setminus \{x\}, \mathcal{I}'')$ where $\mathcal{I}'' = \{T \subseteq S \setminus \{x\} : T \cup \{x\} \in \mathcal{I}\}$ is the matroid obtained by contracting x in M. The rank function of $M/\{x\}$, r_2, is related to the rank function of M by the formula $r_2(T) = r(T \cup \{x\}) - 1$ for $T \subseteq S \setminus x$.*

We now consider the problem of finding a minimum cost basis in a matroid. Given a matroid $M = (S, \mathcal{I})$ and a cost function $c : S \to \mathbb{R}$, the task is to find a basis of M of minimum cost. In the special case of graphic matroids the problem generalizes the minimum spanning tree problem which we studied in Section 3.2. Interestingly, the greedy algorithm gives an efficient algorithm for the minimum cost matroid basis problem [84] and is precisely the structure where greedy algorithm always work [29, 39].

3.4.1 Linear Program

In Figure 3.12 we give a linear programming relaxation $LP_{mat}(M)$ for the minimum cost matroid basis problem given by Edmonds [29]. Edmonds [28, 29, 30] also showed that the linear program is integral.

Solving the linear program. Observe that the above linear program is exponential in size and hence, an efficient separation routine is needed to separate over these constraints. The separation routine needs to check that $x(T) \leq r(T)$ for each $T \subseteq S$. The polynomial time solvability follows from submodular minimization [45]. Also, Cunningham [20] provides a combinatorial algorithm for such a separation routine which as an input uses the independence oracle for matroid $\mathcal{M}$.

We prove the following theorem of Edmonds [29] via the iterative method.

$$\text{maximize} \quad \sum_{e \in S} c_e x_e \tag{3.3}$$

$$\text{subject to} \quad x(S) = \sum_{e \in S} x_e = r(S) \tag{3.4}$$

$$x(T) = \sum_{e \in T} x_e \leq r(T) \qquad \forall T \subseteq S \tag{3.5}$$

$$x_e \geq 0 \qquad \forall e \in S \tag{3.6}$$

Figure 3.12: Linear Program for the Minimum Cost Matroid Basis.

Theorem 3.29 *[29] Every vertex optimal solution to the $LP_{mat}(M)$ is integral.*

Before we prove the theorem, we characterize vertex solutions via a structured set of tight independent constraints. Again the uncrossing method plays a crucial role. The characterization follows from the results of Edmonds [28] and we include the proof outline for completeness.

3.4.2 Characterization of Vertex Solutions

We now give a characterization of the vertex solutions of the linear program $LP_{mat}(M)$ by a small set of tight independent constraints.

We now use an uncrossing argument to show that independent set of tight constraints defining a vertex of $LP_{mat}(M)$ can be chosen to form a chain. Given a vertex solution x of $LP_{mat}(M)$ let $\mathcal{F} = \{T \subseteq S : x(T) = r(T)\}$ be the set of tight constraints. We now show that $\mathcal{F}$ is closed under intersection and union.

Lemma 3.30 *If $U, V \in \mathcal{F}$, then both $U \cap V$ and $U \cup V$ are in $\mathcal{F}$. Furthermore, $\chi(U) + \chi(V) = \chi(U \cap V) + \chi(U \cup V)$.*

Proof:

$$\begin{aligned} r(U) + r(V) &= x(U) + x(V) \\ &= x(U \cap V) + x(U \cup V) \\ &\leq r(U \cap V) + r(U \cup V) \\ &\leq r(U) + r(V). \end{aligned}$$

The first equality is by the fact that $U, V \in \mathcal{F}$. The second equality follows from basic set properties. The third inequality follows from the constraints in the $LP_{mat}(M)$. The last equality is because of properties of rank function r as shown in Lemma 3.26.

Since there are no elements in $U \setminus V$ and $V \setminus U$ in the support of x, we have $\chi(U) + \chi(V) = \chi(U \cap V) + \chi(U \cup V)$. □

Lemma 3.31 *If $\mathcal{L}$ is a maximal chain subfamily of $\mathcal{F}$, then $span(\mathcal{L}) = span(\mathcal{F})$.*

Proof: The proof follows exactly the same argument as in Lemma 3.11. We show exactly where the argument differs and why we obtain a chain in this case while we could only argue a laminar structure in Lemma 3.11.

Lemma 3.30 shows that two tight sets A and B can always be uncrossed and not only when A and B intersect as was the case in Lemma 3.10. Hence, even if A, B are two tight sets and $A \cap B = \emptyset$, we can *uncross* them and ensure that no such two sets exists in family of constraints defining x. □

Thus we have the following characterization given by Edmonds [28].

Lemma 3.32 *Let x be any vertex solution of the linear program $LP_{mat}(M)$. Then there exists a chain $\mathcal{L}$ such that x is the unique solution to the following linear system.*

$$x(T) = r(T) \quad \forall T \in \mathcal{L}.$$

Moreover, the characteristic vectors $\{x_T : T \in \mathcal{L}\}$ are linearly independent and $|\mathcal{L}| = |S|$.

3.4.3 Iterative Algorithm

We now give an iterative algorithm which constructs an integral solution from the optimal vertex solution to linear program $LP_{mat}(M)$. The algorithm is shown in Figure 3.13.

We now show that the iterative algorithm returns an optimal solution. We first show that the algorithm always finds an element with $x_e \in \{0, 1\}$. Then we show using a simple inductive argument that the solution returned by the algorithm is optimal, thus proving Theorem 3.29.

Lemma 3.33 *For any vertex solution x of the $LP_{mat}(M)$ with $x_e > 0$ for every element e, there exists an element e with $x_e = 1$.*

Iterative minimum cost matroid basis algorithm

1. Initialization $B \leftarrow \emptyset$.
2. While B is not a basis
   (a) Find a vertex optimal solution x of the $LP_{mat}(M)$ and delete every element e with $x_e = 0$ from $\mathcal{M}$, i.e., $M \leftarrow M \setminus e$.
   (b) If there is an element e such that $x_e = 1$ then and update $B \leftarrow B \cup \{e\}$, $M \leftarrow M/e$.
3. Return B.

Figure 3.13: Iterative Minimum Cost Matroid Basis Algorithm.

Proof: Suppose for contradiction $0 < x_e < 1$ for each $e \in S$. Then the number of variables is exactly $|S|$. Since there is no element e with $x_e = 0$, every tight inequality is of the form $x(T) = r(T)$. By Lemma 3.31, there are $|\mathcal{L}|$ linearly independent tight constraints of the form $x(T) = r(T)$ for $T \in \mathcal{L}$ where $\mathcal{L}$ is a chain. Since $0 < x_e < 1$ for each element, thus there is set of size one in the chain. Therefore, we have $|\mathcal{L}| \leq |S| - 1$ from Lemma 2.10 which is a contradiction to Lemma 3.32. □

It remains to check that the returned solution is a minimum cost basis, which is proved in the following theorem.

Theorem 3.34 *The Iterative Matroid Basis Algorithm returns a minimum cost basis in polynomial time.*

Proof: This is proved by induction on the number of iterations of the algorithm. The base case is trivial to verify. Let $M = (S, \mathcal{I})$ denote the matroid in the current iteration. If the algorithm finds an element e such that $x_e = 0$ we update the matroid to $M \setminus e$. Observe that x restricted to $S \setminus \{e\}$, say x', is a feasible solution to $LP_{mat}(M \setminus e)$. This is easily checked using the rank function of $M \setminus e$ which is the same as rank function of M by Definition 3.27. By induction, we find a basis B of $M \setminus e$ of cost at most $c \cdot x'$. Observe that B is also a basis of M and costs at most $c \cdot x' = c \cdot x$. Hence, the induction claim is true in this case.

Now, suppose the algorithm selects an element e such that $x_e = 1$. Then the algorithm updates the matroid M to M/e and B to $B \cup \{e\}$. Let r denote the rank function of M and r' denote the rank function of M/e. We now claim that x restricted to

$S \setminus \{e\}$, say x', is a feasible solution to $LP_{mat}(M/e)$. For any set $T \subseteq S \setminus \{x\}$, we have $x'(T) = x(T \cup \{e\}) - x_e = x(T \cup \{e\}) - 1 \leq r(T \cup \{e\}) - 1 = r'(T)$. By the induction hypothesis, we obtain a basis B' of M/e of cost at most $c \cdot x'$. Observe that $B' \cup \{e\}$ is a basis of M by Definition 3.28 and costs at most $c \cdot x' + c(e) = c \cdot x$ as required. This shows that the algorithm returns a minimum cost basis of matroid M. □

3.5 Perfect Matchings in General Graphs

In this section, we consider the problem of finding a minimum cost perfect matching in a graph $G = (V, E)$ with even number of vertices. The minimum cost matching problem in bipartite graph, which we considered in Section 3.1, is considerably simpler than in general graphs. Indeed the linear programming relaxation considered in Section 3.1 for the bipartite matching problem has strictly fractional vertex solutions for general graphs. Edmonds [25] gave a linear programming formulation for the minimum cost perfect matching problem which is integral and the famous primal-dual *Blossom* algorithm which was the first polynomial time algorithm for the problem. The Blossom algorithm also provides a proof of the integrality of the linear program. Subsequently, other proofs of integrality were given including, finding a decomposition of a fractional solution into convex combination of matchings [3, 92], characterizing all the facets of the matching polytope [4], [51]. We refer the reader to the text by Lovász and Plummer [72] for details about the matching problem.

In this section, we will show the integrality of the linear program given by Edmonds [25] using the iterative method.

3.5.1 Linear Program

Given a graph $G = (V, E)$ with an even number of vertices and a cost function $c : E \rightarrow \mathcal{R}_+$, the linear programming relaxation for the perfect matching problem is given by the following $LP_M(G)$.

$$\text{minimize} \quad c(x) = \sum_{e \in E} c_e x_e \tag{3.7}$$

$$\text{subject to} \quad \sum_{e \in \delta(v)} x_e = 1 \qquad \forall v \in V \tag{3.8}$$

$$\text{subject to} \quad \sum_{e \in \delta(S)} x_e \geq 1 \qquad \forall S \subset V, |S| \text{ odd}, \tag{3.9}$$

$$x_e \geq 0 \qquad \forall e \in E \tag{3.10}$$

Observe that if G is not bipartite then the above linear program is not integral if we do not include the *odd-set* inequalities (3.9) as shown in Figure 3.14.

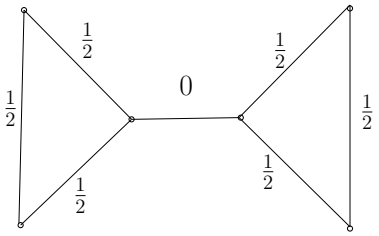


Figure 3.14: In the above figure, if the cost of the cut edge is large, then the all-half solution on the triangles is an optimal solution to the linear program with degree constraints. Thus the odd set inequalities that cut such solutions off are necessary.

We prove the following theorem showing integrality of the linear program.

Theorem 3.35 *[25] Every vertex optimal solution to the linear program $LP_M(G)$ is integral.*

We give an iterative algorithm proving Theorem 3.35 but first we give a characterization of a vertex solution of $LP_M(G)$.

3.5.2 Characterization of Vertex Solutions

We prove the following crucial lemma characterizing vertex solutions. Again the uncrossing technique and the rank lemma forms the basis of the argument. The characterization follows directly from the results of Cunningham and Marsh [21]. We include the proofs here for completeness.

Lemma 3.36 *Given any vertex solution x of $LP_M(G)$ let the set of tight constraints $\tau = \{S : x(\delta(S)) = 1, S \subset V, |S| \text{ odd}\}$. Then there exists a laminar family $\mathcal{L} \subseteq \tau$ such that*

1. *$\chi(\delta(S))$ for $S \in \mathcal{L}$ are linearly independent vectors.*
2. *$G[S]$ is connected for each set $S \in \mathcal{L}$.*
3. *$span(\{\chi(\delta(S)) : S \in \mathcal{L}\}) = span(\{\chi(\delta(S)) : S \in \tau\})$.*

Before we give the proof of Lemma 3.36, we first prove the following claims.

Lemma 3.37 *If $A, B \in \tau$ and $A \cap B \neq \emptyset$, then one of the following is true*

1. *$A \cap B \in \tau$, $A \cup B \in \tau$ and $\chi(\delta(A)) + \chi(\delta(B)) = \chi(\delta(A \cap B)) + \chi(\delta(A \cup B))$*
2. *$A \setminus B \in \tau$, $B \setminus A \in \tau$ and $\chi(\delta(A)) + \chi(\delta(B)) = \chi(\delta(A \setminus B)) + \chi(\delta(B \setminus A))$*

Proof: Let $A, B \in \tau$. First assume $A \cap B$ is odd. Then $A \cup B$ is also odd. Hence, we have

$$2 = 1 + 1 \leq x(\delta(A \cap B)) + x(\delta(A \cup B)) \leq x(\delta(A)) + x(\delta(B)) = 2$$

Hence, all inequalities are satisfied at equality implying that all cross-edges between A and B have a value of 0 in x (and hence not present in the support). Therefore, $A \cap B \in \tau$, $A \cup B \in \tau$ and $\chi(\delta(A)) + \chi(\delta(B)) = \chi(\delta(A \cap B)) + \chi(\delta(A \cup B))$.

A similar argument shows that second case holds when $A \cap B$ is even in which case $|A \setminus B|$ and $|B \setminus A|$ are odd. □

Claim 3.38 *If $S \in \tau$ such that $G[S]$ is not connected then there exists $R \subseteq S$ such that $R \in \tau$ and $\delta(R) = \delta(S)$.*

Proof: Let $S \in \tau$ be a set such that $G[S]$ is not connected. The let $R_1, \ldots, R_k$ be the connected components of $G[S]$ where R_1 is of odd cardinality (such a component must exist as $|S|$ is odd). Now, we have $\delta(R_1) \subseteq \delta(S)$ but $1 \leq x(\delta(R_1)) \leq x(\delta(S)) = 1$. Hence, equality must hold everywhere and in the support graph, we have $\delta(R_1) = \delta(S)$. □

Now we prove Lemma 3.36.

Proof of Lemma 3.36: From the Rank lemma, it follows that any set of maximally independent tight constraints satisfies conditions (1) and (3). Claim 3.38 shows that it is

enough to concentrate on the tight constraints for odd sets which are connected. Let $\tau' = \{S \in \tau : G[S] \text{ is connected}\}$. Then Claim 3.38 implies that $span(\{\chi(\delta(S)) : S \in \tau'\}) = span(\{\chi(\delta(S)) : S \in \tau\})$.

We now show that any maximal laminar family in τ' is indeed a maximal set of tight independent constraints proving the lemma. Let $\mathcal{L}$ be a maximal independent laminar family of τ'. We claim that $\mathcal{L}$ satisfies the properties of Claim 3.36. It is enough to show that sets in $\mathcal{L}$ span all sets in τ'. Assume that $\chi(\delta(S)) \notin span(\mathcal{L})$ for some $S \in \tau'$. Choose one such set S that intersects as few sets of $\mathcal{L}$ as possible. Since $\mathcal{L}$ is a maximal laminar family, there exists $T \in \mathcal{L}$ that intersects S. From Lemma 3.37, we have that $S \cap T$ and $S \cup T$ are also in τ or $S \setminus T$ and $T \setminus S$ are in τ. Assume that we have the first case and thus we have $\chi(\delta(S)) + \chi(\delta(T)) = \chi(\delta(S \cap T)) + \chi(\delta(S \cup T))$. Since $\chi(\delta(S)) \notin span(\mathcal{L})$, either $\chi(\delta(S \cap T)) \notin span(\mathcal{L})$ or $\chi(\delta(S \cup T)) \notin span(\mathcal{L})$. In either case, we have a contradiction because both $S \cup T$ and $S \cap T$ intersect fewer sets in $\mathcal{L}$ than S; this is because every set that intersects $S \cup T$ or $S \cap T$ also intersects S. In the other case, we have a similar argument showing a contradiction. ■

3.5.3 Iterative Algorithm

The following is a simple iterative procedure which returns a matching of optimal cost and also shows that the above linear program is integral. The proof is in two parts. First

Iterative Matching Algorithm

1. Initialization $F \leftarrow \emptyset$.
2. While $V(G) \neq \emptyset$
   (a) Find a vertex optimal solution x of $LP_M(G)$ and remove every edge e with $x_e = 0$ from G.
   (b) If there is an edge $e = \{u, v\}$ such that $x_e = 1$ then update $F \leftarrow F \cup \{e\}$, $G \leftarrow G \setminus \{u, v\}$.
3. Return F.

Figure 3.15: Iterative Matching Algorithm.

assume that we can always find an edge e with $x_e = 1$ in Step 2b of the algorithm. We show that the solution returned F is a matching of G of cost no more than the initial LP solution x, and hence it is also a minimum cost matching. Then we show that the algorithm indeed finds an edge with $x_e = 1$. The proof of the following claim is identical

to proof of Claim 3.4 and is omitted.

Claim 3.39 *Assuming that the iterative algorithm finds an edge e such that $x_e \in \{0,1\}$ then the algorithm returns an optimal matching.*

It remains to show that the algorithm always finds an edge e such that $x_e = 1$ in Step 2b.

Lemma 3.40 *Given any vertex solution x of $LP_M(G)$ there must exist an edge e such that $x_e = 0$ or $x_e = 1$.*

Proof: Suppose for sake of contradiction $0 < x_e < 1$ for each edge $e \in E$. Let $\mathcal{L}$ be the laminar family given by Claim 3.36. We show a contradiction by showing that $|E| > |\mathcal{L}|$. This is done by a token argument. Initially we give two tokens for each edge and which gives one each to its endpoint for a total of $2|E|$ tokens. Now, we collect two tokens for each member in the laminar family and at least one extra token for total of $2|\mathcal{L}|+1$ tokens giving us the desired contradiction. The token redistribution is done by an inductive argument.

Claim 3.41 *For any $S \in \mathcal{L}$, using tokens for the vertices in S, we can give two tokens to each set in $\mathcal{L}$ in the subtree rooted at S and $|\delta(S)|$ tokens to S.*

Proof: The proof is by induction on the height of the subtree rooted at S.

Base Case. S is a leaf. S can take $|\delta(S)|$ tokens for one for each edge incident at some vertex in S.

Induction Case. Let S have children $R_1, R_2, \ldots, R_k$ (where some of the R_i could be singletons). From induction hypothesis R_i receives $|\delta(R_i)|$ tokens. We use these tokens, one for each edge in $\delta(R_i)$, and the tokens assigned to S to give two tokens to each R_i and $|\delta(S)|$ tokens to S.

From Lemma 3.36 we have that $G[S]$ is connected. Let H be the connected graph formed from $G[S]$ by contracting each R_i in to a singleton vertex. If H has at least k edges then, we use two tokens for each such edge to give two tokens to the children. We still have $|\delta(S)|$ tokens left, one for each edge in $\delta(S)$.

Else, we must have that H has exactly k vertices, fewer than k edges and is connected. Hence, H must be a tree. As every tree is a bipartite graph, let H_1 and H_2 be the bipartition. Now, we prove a contradiction to independence of constraints. Let $|H_1| > |H_2|$

(cardinality of S is odd and therefore the number of vertices in H is odd). Let $A = \cup_{R_i \in H_1} \delta(R_i)$ and $B = \cup_{R_i \in H_2} \delta(R_2)$. Then $\delta(S) = A \Delta B$. But $x(A) = \sum_{R_i \in H_1} x(\delta(R_i)) = |H_1|$ and similarly, $x(B) = |H_2|$. Thus, we have $x(A \setminus B) \geq |H_1| - |H_2| \geq 1$. But, then we have $1 = x(\delta(S)) = x(A \Delta B) = x(A \setminus B) + x(B \setminus A) \geq 1$ implying that $x(B \setminus A) = 0$ and $B \setminus A = \emptyset$. But then the constraints for S and its children in H_1 are dependent since $x(\delta(S)) = x(A) - x(B)$. A contradiction. □

We apply Claim 3.41 to each root of every tree in $\mathcal{L}$. If $|\delta(S)| \geq 3$ for any root S or there is a vertex not in any root then we have an extra token and Lemma 3.40 holds. Else, the graph H formed by contracting each root set of $\mathcal{L}$ in to a singleton vertex is a cycle cover. If the cycle cover contains an even cycle C then the then the constraints for odd sets in C are dependent which is a contradiction. Else, if there is an odd cycle C let S_C denote the union of sets in C. Observe that S_C is disjoint union of odd number of odd cardinality sets and therefore $|S_C|$ is odd. But $|\delta(S_C)| = 0$ which contradicts the inequality $x(\delta(S_C)) \geq 1$. □

Thus we prove Theorem 3.35 showing the integrality of the linear programming relaxation $LP_M(G)$ for the perfect matching problem in general graphs.

4

Minimum Bounded-Degree Spanning Trees

In this chapter we study the MINIMUM BOUNDED-DEGREE SPANNING TREE (MBDST) problem. Recall that in an instance of the MBDST problem we are given an undirected graph $G = (V, E)$, edge costs given by $c : E \to \mathbb{R}$, degree bound $B_v \geq 1$ for each $v \in V$ and the task is to find a spanning tree of minimum cost which satisfies the degree bounds. We prove the following theorem.

Theorem 4.1 *There exists a polynomial time algorithm which given an instance of the MBDST problem returns a spanning tree T such that $deg_T(v) \leq B_v + 1$ and cost of the tree T is smaller than the cost of any tree which satisfies the degree bounds.*

We prove Theorem 4.1 using the iterative relaxation technique. We first prove a weaker guarantee where the degree bound is violated by an additive amount of two in Section 4.1. This matches the result of Goemans [42] which was the previous best result for this problem. The proof in this case is simpler and will illustrate the iterative method. In section 4.2 we give an algorithm where the degree bound is violated by an additive amount of one using a fractional token argument. In section 4.3 we consider the generalization when both upper and lower degree bounds are present and give an algorithm which returns a tree of optimal cost which violates the degree bounds by at most an additive amount of one.

4.1 An Additive 2 Approximation Algorithm

In this section we first present an $(1, B_v + 2)$-approximation algorithm for the MBDST problem via iterative rounding. This algorithm is simple, and it illustrates the idea of iterative relaxation by removing degree constraints.

4.1.1 Linear Programming Relaxation

We use the following standard linear programming relaxation for the MBDST problem, which we denote by LP-MBDST(G, $\mathcal{B}, W$). In the following we assume that degree bounds are given for vertices only in a subset $W \subseteq V$. Let $\mathcal{B}$ denote the vector of all degree bounds B_v, one for each $v \in W$.

$$\text{minimize} \quad c(x) = \sum_{e \in E} c_e x_e \tag{4.1}$$

$$\text{subject to} \quad x(E(V)) = |V| - 1 \tag{4.2}$$

$$x(E(S)) \leq |S| - 1 \qquad \forall S \subset V \tag{4.3}$$

$$x(\delta(v)) \leq B_v \qquad \forall v \in W \tag{4.4}$$

$$x_e \geq 0 \qquad \forall e \in E \tag{4.5}$$

Separation over the inequalities in the above linear program is in polynomial time and follows from Theorem 3.8. An alternative is to write a compact formulation for the above linear program [107] which has polynomially many variables and constraints.

4.1.2 Characterization of Vertex Solutions

We first give a characterization of a vertex solution of LP-MBDST($G, \mathcal{B}, W$). We remove all edges with $x_e = 0$ and focus only on the support of the vertex solution and the tight constraints from (4.2)-(4.4). Let $\mathcal{F} = \{S \subseteq V : x(E(S)) = |S| - 1\}$ be the set of tight constraints from (4.2)-(4.3). From an application of Rank Lemma and the characterization of vertex solutions to the spanning tree polyhedron in Lemma 3.13 we get the following.

Lemma 4.2 *Let x be any vertex solution of LP-MBDST($G, \mathcal{B}, W$) with $x_e > 0$ for each edge $e \in E$. Then there exists a set $T \subseteq W$ with $x(\delta(v)) = B_v$ for each $v \in T$ and a laminar family $\mathcal{L} \subseteq \mathcal{F}$ such that*

1. *The vectors $\{\chi(E(S)) : S \in \mathcal{L}\} \cup \{\chi(\delta(v)) : v \in T\}$ are linearly independent.*
2. *The vector space generated by $\{\chi(\delta(v)) : v \in T$ and span($\mathcal{L}$)=span($\mathcal{F}$).*
3. *$|\mathcal{L}| + |T| = |E|$.*

4.1.3 Iterative Algorithm I

In this section, we give an iterative algorithm which returns a tree of optimal cost and violates the degree bound within an additive error of two. The algorithm is given in Figure 4.1.

MBDST Algorithm

1. Initialization $F \leftarrow \emptyset$.
2. While $V(G) \neq \emptyset$
   (a) Find a vertex optimal solution x of LP-MBDST$(G, \mathcal{B}, W)$ and remove every edge e with $x_e = 0$ from G. Let the support of x be E.
   (b) If there exists a vertex $v \in V$, such that there is at most one edge $e = uv$ incident at v in E, then update $F \leftarrow F \cup \{e\}$, $G \leftarrow G \setminus \{v\}$, $W \leftarrow W \setminus \{v\}$, and also update $\mathcal{B}$ by setting $B_u \leftarrow B_u - 1$.
   (c) If there exists a vertex $v \in W$ such that $deg_E(v) \leq 3$ then update $W \leftarrow W \setminus \{v\}$.
3. Return F.

Figure 4.1: MBDST Algorithm.

In the next lemma we prove (by a very simple counting argument) that in each iteration we can proceed by applying either Step 2b or Step 2c; this will ensure that the algorithm terminates.

Lemma 4.3 *Any vertex solution x of LP-MBDST$(G, \mathcal{B}, W)$ with $x_e > 0$ for each edge $e \in E$ must satisfy one of the following.*

(a) There is a vertex $v \in V$ such that $deg_E(v) = 1$.

(b) There is a vertex $v \in W$ such that $deg_E(v) \leq 3$.

Proof: Suppose for sake of contradiction that both (a) and (b) are not satisfied. Then every vertex has at least two edges incident to it and every vertex in W has at least four edges incident at it. Therefore, $|E| \geq (2(n-|W|)+4|W|)/2 = n+|W|$, where $n = |V(G)|$.

By Lemma 4.2, there is a laminar family $\mathcal{L}$ and a set $T \subseteq W$ of vertices such that $|E| = |\mathcal{L}|+|T|$. Since $\mathcal{L}$ is a laminar family which only contains subsets of size at least two, from Proposition 2.10 we have $|\mathcal{L}| \leq n-1$. Hence, $|E| = |\mathcal{L}|+|T| \leq n-1+|T| \leq n-1+|W|$, a contradiction. □

We now prove the correctness of the algorithm.

Theorem 4.4 *The iterative algorithm in Figure 4.1 returns a tree T of optimal cost such that $deg_T(v) \leq B_v + 2$ for each $v \in V$.*

Proof: The proof that the cost of tree returned is at most the cost of the linear programming solution is identical to the proof of Theorem 3.7.

We show that the degree of any vertex v is at most $B_v + 2$. At any iteration, let F denote the set of edges selected and let B'_v denote the current residual degree bound of v.

Claim 4.5 *While the degree constraint of v is present, $deg_F(v) + B'_v = B_v$.*

Proof: The proof is by induction on the number of iterations of the algorithm. Initially, $F = \phi$ and $B'_v = B_v$ and the claim holds. At any iteration, whenever we include an edge $e \in \delta(v)$ in F, we reduce B'_v by one and hence the equality holds true. □

When the degree bound for the vertex v is removed then at most 3 edges are incident at v and $B'_v \neq 0$. In the worst case, we may select all three edges in the solution. Hence,

$$deg_T(v) \leq B_v - B'_v + 3 \leq B_v + 2$$

where $B'_v \geq 1$ is the degree bound of v when the degree constraint is removed. □

4.2 An Additive 1 Approximation Algorithm

In this section, we give an iterative algorithm which returns a tree of optimal cost and violates the degree bound within an additive error of one proving Theorem 4.1. The algorithm is given in Figure 4.2

The algorithm proceeds as follows. The algorithm maintains a subset W of vertices on which it places a degree bound. In each iteration, the algorithm finds a vertex $v \in W$ such that the degree of v in the support is at most $B_v + 1$ and removes the degree constraint for v. Observe that once all the degree constraints are removed we obtain the linear program for the minimum spanning tree problem which we showed in Section 3.2 is integral. Hence, when $W = \emptyset$ and the algorithm terminates and returns a tree.

MBDST Algorithm

1. While $W \neq \emptyset$
   (a) Find a vertex optimal solution x of LP-MBDST$(G, \mathcal{B}, W)$ and remove every edge e with $x_e = 0$ from G. Let the support of x be E.
   (b) If there exists a vertex $v \in W$ such that $deg_E(v) \leq B_v + 1$ then update $W \leftarrow W \setminus \{v\}$.
2. Return E.

Figure 4.2: Additive +1 MBDST Algorithm.

At each step we only relax the linear program. Hence, the cost of the final solution is at most the cost of the initial linear programming solution. Thus the tree returned by the algorithm has optimal cost. A simple inductive argument as in proof of Theorem 4.4 also shows that the degree bound is violated by at most an additive one. The degree bound is violated only when we remove the degree constraint and then $deg_E(v) \leq B_v + 1$. Thus, in the worst case, if we include all the edges incident at v in T, degree bound of v is violated by at most an additive one. Thus we have the following lemma.

Lemma 4.6 *If in each iteration, the algorithm finds a vertex to remove a degree bound for some vertex $v \in W$ and terminates when $W = \emptyset$ then the algorithm returns a tree T of optimal cost and $deg_T(v) \leq B_v + 1$ for each $v \in V$.*

It remains to show that the iterative relaxation algorithm finds a degree constraint to remove at each step. From Lemma 4.2 we have that there exists a laminar family $\mathcal{L} \subseteq \mathcal{F}$ and $T \subseteq W$ such that $|\mathcal{L}|+|T| = |E|$ and constraints for sets in $\mathcal{L}$ are linearly independent. Observe that if $T = \phi$ then only the spanning tree inequalities define the solution x. Hence, x must be integral by Theorem 3.7. In the other case, we show that there must be a vertex in W whose degree constraint can be removed.

Lemma 4.7 *Let x be a vertex solution to LP-MBDST$(G, \mathcal{B}, W)$ such that $x_e > 0$ for each $e \in E$. Let $\mathcal{L}$ and $T \subseteq W$ correspond to the tight set constraints and tight degree constraints defining x as given by Lemma 4.2. If $T \neq \phi$ then there exists some vertex $v \in W$ such that $deg_E(v) \leq B_v + 1$.*

Proof: We present a simple proof given by Bansal et al. [5] based on the a fractional token argument. The proof will build on the second proof of Lemma 3.16. Suppose for the sake of contradiction, we have $T \neq \phi$ and $deg_E(v) \geq B_v + 2$ for each $v \in W$.

Claim 4.8 *We can assume that $\chi(e) \in span(\mathcal{L})$ for each e such that $x_e = 1$.*

Proof: Since Lemma 4.2 holds for any maximal laminar family we construct one laminar family $\mathcal{L}'$ such that $\chi(e) \in span(\mathcal{L}')$. Extending $\mathcal{L}'$ to a maximal laminar family $\mathcal{L}$ will give the desired property. Let $E_t = \{e \in E : x_e = 1\}$ and C be any connected component of the graph induced by E_t. Order the edges in $E(C) \cap E_t = \{e_1, \ldots, e_r\}$ such that the graph induced by $\{e_1, \ldots, e_i\}$ is connected for each $1 \leq i \leq r$. Such an ordering exists since C is connected. Include the set of vertices spanned by $C_i = \{e_1, \ldots, e_i\}$ in $\mathcal{L}'$. Such a set is tight since $x(C_i) = i$ and C_i contains $i + 1$ vertices. Moreover the sets $\{C_i : 1 \leq i \leq r\}$ form a laminar family (actually a chain). Since, $\chi(e_i) = \chi(C_i) - \chi(C_{i-1})$ for each $1 \leq i \leq r$ where $C_0 = \phi$, $\chi(e) \in span(\{\chi(C_i) : 1 \leq i \leq r\}$ for each $e \in E(C)$. We repeat this argument for each connected component of E_t and include the corresponding chains in $\mathcal{L}'$. Since the connected components are over disjoint set of vertices, the sets included do not intersect and the collection of sets form a laminar family proving the claim. □

We now show a contradiction by a token argument. We give one token for each edge in E. We then redistribute the token such that each vertex in T and each set in $\mathcal{L}$ gets one token and we still have extra tokens left. This will contradict $|E| = |T| + |\mathcal{L}|$ (Lemma 4.2). The token redistribution is as follows. Each edge $e \in E$ gives $\frac{1-x_e}{2}$ to each of its endpoints for the degree constraints and x_e token to the smallest set in $\mathcal{L}$ containing both endpoints of e.

We now show that each vertex with a tight degree constraint gets one token. Let $v \in T$ be such a vertex. Then v receives $\frac{1-x_e}{2}$ tokens for each edge incident at v for a total of

$$\sum_{e \in \delta(v)} \frac{1 - x_e}{2} = \frac{deg_E(v) - B_v}{2} \geq 1$$

where the first equality holds since $\sum_{e \in \delta(v)} x_e = B_v$ and the inequality holds since $deg_E(v) \geq B_v + 2$.

Now we show that each member $S \in \mathcal{L}$ also obtains one token. S receives x_e token for each edge e such that S is the smallest set containing both endpoints of e. Let $R_1, \ldots, R_k$

be the children of S in the laminar family $\mathcal{L}$ where $k \geq 0$. We have

$$\begin{aligned} x(E(S)) &= |S| - 1 \\ x(E(R_i)) &= |R_i| - 1 \text{ for each } 1 \leq i \leq k \\ \implies x(E(S)) - \sum_{i=1}^{k} x(E(R_i)) &= |S| - 1 - \sum_{i=1}^{k} (|R|_i - 1) \\ \implies x(A) &= |S| - 1 - \sum_{i=1}^{k} (|R_i| - 1) \end{aligned}$$

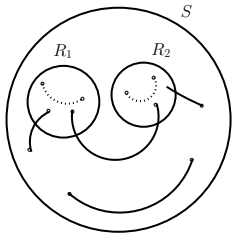


Figure 4.3: The solid edges are in A while the dotted edges are not.

where $A = E(S) \setminus (\cup_{i=1}^{k} E(R_i))$ (see Figure 4.3). Observe that S receives exactly $x(A)$ tokens which is an integer by the above equation. But if $x(A) = 0$ then $\chi(E(S)) = \sum_{i=1}^{k} \chi(E(R_i))$ which contradicts the independence. Hence, each set also receives at least one token.

Now, we argue that there is an extra non-zero token left for contradiction of Lemma 4.2. If $V \notin \mathcal{L}$ then there exists an edge e which is not contained in any set of $\mathcal{L}$ and the x_e token for that edge gives us the contradiction. Similarly, if there is a vertex $v \in W \setminus T$ then v also collects one extra token and we get the desired contradiction. Moreover, if there is a vertex $v \in V \setminus T$ then each edge e incident at v must have $x_e = 1$ else $\frac{1-x_e}{2} > 0$ tokens are extra giving a contradiction. But then $\chi(\delta(v)) \in \mathit{span}(\mathcal{L})$ for each $v \in V \setminus T$ since $\chi(e) \in \mathit{span}(\mathcal{L})$ for each $e \in \delta(v)$ from Claim 4.8. But we have

$$\sum_{v \in V} \chi(\delta(v)) = 2x(E(V))$$

where each of the term on the left side is either $\chi(\delta(v))$ for $v \in T$ or in $\mathit{span}(\mathcal{L})$. But this is a linear dependence in the set of all tight independent constraints defining the vertex solution x giving a contradiction. □

This completes the proof of Theorem 4.1.

4.3 An Additive ± 1 Approximation Algorithm

In this section, we consider the MBDST problem when both lower degree bounds A_v and upper degree bounds B_v are given. We present an approximation algorithm which returns a tree T of optimal cost and $A_v - 1 \leq deg_T(v) \leq B_v + 1$ for each $v \in V$. We actually present an algorithm for a more general problem, the Minimum Bounded-Degree Connecting Tree (MBDCT) problem.

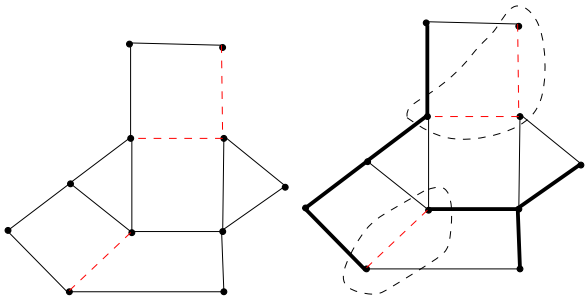
Connecting Tree Problem

The Minimum Bounded-Degree Connecting Tree problem is defined as follows. We are given a graph $G = (V, E)$, degree lower degree bounds A_v for each vertex in $U \subseteq V$ and upper degree bounds B_v for each vertex v in some subset $W \subseteq V$, a cost function $c : E \to \mathbb{R}$, and a forest F on V. We assume without loss of generality that $E(F) \cap E(G) = \emptyset$. The task is to find a minimum cost forest H such that $H \cup F$ is a spanning tree of G and $A_v \leq d_H(v) \leq B_v$. We call such a forest H an *F-tree* of G, and a connected component of F a *supernode*; note that an isolated vertex of F is also a supernode. Intuitively, the forest F is the partial solution we have constructed so far, and H is a spanning tree in the graph where each supernode is contracted into a single vertex. We denote this contracted graph by G/F. Formally $V(G/F) =$ connected components of F and $E(G/F) =$ edges between different components of F. Observe that when $E(F) = \emptyset$ the MBDCT problem is just the MBDST problem.

4.3.1 Linear Program

We need some notation to define the linear programming relaxation for the MBDCT problem. For any set $S \subseteq V(G)$ and a forest F on G, let $F(S)$ be the set of edges in F with both endpoints in S, i.e., $\{e \in F : |e \cap S| = 2\}$. Note that $F(V)$ is just equal to $E(F)$. We denote $\mathcal{C}(F)$ the sets of supernodes of F. A set S is *non-intersecting with* F if for each $C \in \mathcal{C}(F)$ we either have $C \subseteq S$ or $C \cap S = \emptyset$. We denote $\mathcal{I}(F)$ the family of all subsets which are non-intersecting with F.

We assume the lower bounds are given on a subset of vertices $U \subseteq V$ and upper bounds on a subset $W \subseteq V$. Let $\mathcal{A}$ ($\mathcal{B}$) denote the vector of all lower (upper) degree



(a) **(b)**

Figure 4.4: In Figure (a), the dashed edges correspond to F. In Figure (b), the bold edges H form an F-tree of G as $F \cup H$ is a spanning tree of G or equivalently, H is a spanning tree of G/F.

bounds A_v (B_v) for each $v \in U$ ($v \in W$). The following is a linear programming relaxation for the MBDCT problem which we denote by LP-MBDCT$(G, \mathcal{A}, \mathcal{B}, U, W, F)$. In the linear program we have a variable x_e for each edge e which has at most one endpoint in any one component of forest F. Indeed we assume (without loss of generality) that E does not contain any edge with both endpoints in the same component of F.

$$
\begin{aligned}
(4.6) \qquad & \text{minimize} && c(x) &&= && \sum_{e \in E} c_e\, x_e && \\
(4.7) \qquad & \text{subject to} && x(E(V)) &&= && |V| - |F(V)| - 1 && \\
(4.8) \qquad & && x(E(S)) &&\leq && |S| - |F(S)| - 1 && \forall\, S \in \mathcal{I}(F) \\
(4.9) \qquad & && x(\delta(v)) &&\geq && A_v && \forall\, v \in U \\
(4.10) \qquad & && x(\delta(v)) &&\leq && B_v && \forall\, v \in W \\
(4.11) \qquad & && x_e &&\geq && 0 && \forall\, e \in E
\end{aligned}
$$

In the linear program, the constraints from (4.7)-(4.8) and (4.11) are exactly the spanning tree constraints for the graph G/F, the graph formed by contracting each component/supernode of F into a singleton vertex. The constraints from (4.9)-(4.10) are the degree constraints for vertices in U and W. Hence, from the Theorem 3.8, it follows that we can optimize over LP-MBDCT$(G, \mathcal{A}, \mathcal{B}, U, W, F)$ using the ellipsoid algorithm in polynomial time.

4.3.2 Characterization of Vertex Solutions

We give a characterization result for any vertex solution to LP-MBDCT$(G, \mathcal{A}, \mathcal{B}, W, F)$. The proof of the following is straightforward from Rank Lemma and Lemma 3.13 applied to the spanning tree linear program for the graph G/F.

Lemma 4.9 *Let x be any vertex solution of LP-MBDCT$(G, \mathcal{A}, \mathcal{B}, U, W, F)$ such that $x_e > 0$ for each edge $e \in E$ and let $\mathcal{F} = \{S \in \mathcal{I}(F) : x(E(S)) = |S| - |F(S)| - 1\} \bigcup \{\delta(v) : x(\delta(v)) = A_v : v \in U\} \bigcup \{\delta(v) : x(\delta(v)) = B_v : v \in W\}$. Then there exists a set $T_U \subseteq U$, $T_W \subseteq W$ and a laminar family $\emptyset \neq \mathcal{L} \subseteq \mathcal{I}(F)$ such that*

1. *The vectors $\{\chi(E(S)) : S \in \mathcal{L}\} \cup \{\chi(\delta(v)) : v \in T_U \cup T_W\}$ are linearly independent.*
2. *span$(\mathcal{L} \cup \{\delta(v) : v \in T_U \cup T_W\})$=span$(\mathcal{F})$.*
3. *$|E| = |\mathcal{L}| + |T_U| + |T_W|$*

4.3.3 Iterative Algorithm

We now give an iterative algorithm for the MBDCT problem in Figure 4.5.

MBDCT$(G, \mathcal{A}, \mathcal{B}, U, W, F)$

1. If F is a spanning tree then return $\emptyset$ else let $\hat{F} \leftarrow \emptyset$.
2. Find a vertex optimal solution x of LP-MBDCT$(G, \mathcal{A}, \mathcal{B}, U, W, F)$ and remove every edge e with $x_e = 0$ from G.
3. If there exists an edge $e = \{u, v\}$ such that $x_e = 1$ then $\hat{F} \leftarrow \{e\}$, $F \leftarrow F \cup \{e\}$ and $G \leftarrow G \setminus \{e\}$. Also update $\mathcal{A}, \mathcal{B}$ by setting $A_u \leftarrow A_u - 1, B_u \leftarrow B_u - 1$ and $A_v \leftarrow A_v - 1, B_v \leftarrow B_v - 1$.
4. If there exists a vertex $v \in U \cup W$ of degree two, then update $U \leftarrow U \setminus \{v\}$ and $W \leftarrow W \setminus \{v\}$.
5. Return $\hat{F} \bigcup$ MBDCT$(G, \mathcal{A}, \mathcal{B}, U, W, F)$.

Figure 4.5: Connecting Tree Algorithm MBDCT.

For the correctness of the algorithm MBDCT, we shall prove the following key lemma, which will ensure that the algorithm terminates.

Lemma 4.10 *A vertex solution x of LP-MBDCT$(G, \mathcal{A}, \mathcal{B}, U, W, F)$ with support E must satisfy one of the following.*

(a) There is an edge e such that $x_e = 1$.

(b) There is a vertex $v \in U \cup W$ such that $deg_E(v) = 2$.

In MBDCT Algorithm 2, we only remove a degree constraint on $v \in U \cup W$ if v is of degree 2 and there is no 1-edge. Since there is no 1-edge, we must have $A_v \leq 1$. If $v \in U$, then the worst case is $A_v = 1$ but both edges incident at v are not picked in later iterations. If $v \in W$, then the worst case is $B_v = 1$ but both edges incident at v are picked in later iterations. In either case, the degree bound is off by at most 1. Following the same approach in Theorem 4.4, we have the following theorem.

Theorem 4.11 *There is a polynomial time algorithm for the* Minimum Bounded-Degree Connecting Tree *problem which returns a tree T such that $c(T) \leq c \cdot x$ and $deg_T(v) \leq B_v + 1$ for each $v \in W$ and $deg_T(v) \geq A_v - 1$ for each $v \in U$ where x is the optimal solution to the linear program LP-MBDCT$(G, \mathcal{A}, \mathcal{B}, U, W, F)$.*

A Counting Argument

Now we are ready to prove Lemma 4.10. Let $\mathcal{L}$ be the laminar family and $T := T_U \cup T_W$ be the vertices defining the solution x as in Lemma 4.9. Suppose that both (a) and (b) of Lemma 4.10 are not satisfied. We shall derive that $|\mathcal{L}| + |T| < |E|$, which will contradict Lemma 4.9 and complete the proof. We will give two tokens to each edge and collect two tokens for each set in $\mathcal{L}$ and for each vertex in T plus an extra token deriving the contradiction. We do this by induction on the tree representing $\mathcal{L}$.

We call a vertex active if it has degree at least one in the support E. Each component of F is called a supernode. Notice that if a supernode C has only one active vertex v, we can contract C into a single vertex c, set $A_c := A_v$ and $B_c := B_v$, and set $c \in U \iff v \in U$, and set $c \in W \iff v \in W$. Henceforth, we call a supernode which is not a single vertex a *nontrivial supernode*. Hence a non-trivial supernode has at least 2 active vertices. We also have $deg_E(v) \geq 3$ for each $v \in T$.

Each edge receives two tokens which it distributes one each of its endpoints. Hence, each active vertex $v \in V$ receives one token for each edge incident at v. Observe that in the initial assignment each active vertex has at least one excess token, and so a nontrivial supernode has at least two excess tokens. For a vertex v with only one excess token, if

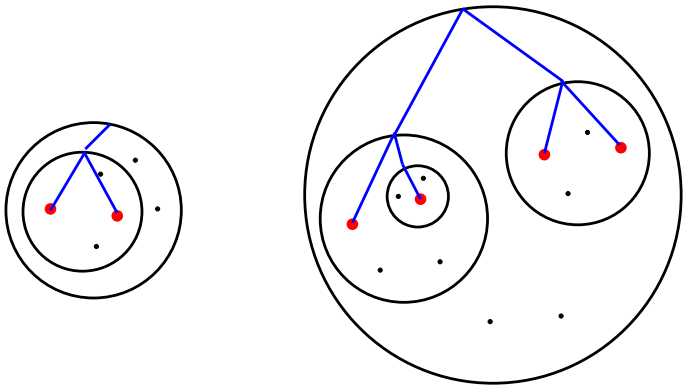


Figure 4.6: In the above figure, the red vertices denote the vertices in T. The blue forest corresponds to the forest for laminar family $\mathcal{L}$ and vertices in T.

$v \notin T$, then v is a degree 1 vertex; if $v \in T$, then v is of degree 3 and $B_v = 1$ or $B_v = 2$.

Suppose every vertex v which is active (and hence has excess tokens) gives all its excess tokens to the supernode it is contained in. We say the number of excess tokens of a supernode is the sum of excess tokens of active vertices in that supernode. Observe that the excess of any supernode is at least one as every supernode has at least one active vertex and each active vertex has at least one excess token.

We call a supernode *special* if its excess is exactly one.

Claim 4.12 *A supernode C is special only if it contains exactly one active vertex $v \in T$ and $deg_E(v) = 3$.*

Proof: If the supernode C has two or more active vertices then the excess of C is at least two. Hence, it must contain exactly one active vertex with exactly one excess token. Also, there must be at least two edges incident at the supernode as $x(\delta(C)) \geq 1$ is a valid inequality. Hence, $deg_E(C) \geq 2$. If $v \notin T$, then both v and thus C will have at least two excess tokens. This implies $v \in T$ and $deg_E(v) = 3$. □

The induction strategy to reach the contradiction works bottom up in $\mathcal{L}$ (vertices in T are leaves) assigning two tokens to each set and three or four tokens to the root of the subtree depending on its features. We describe this next. We contract a special supernode into a single vertex because it contains only one active vertex. Hence, the only special supernodes are singleton vertices in T with degree exactly three. Special vertices with degree bounds at most 2 need careful analysis because some node $S \in \mathcal{L}$ may now only get

three tokens. The following definition gives a characterization of those sets which only get three tokens.

Definition 4.13 *A set $S \neq V$ is* special *if:*

1. $|\delta(S)| = 3$;
2. $x(\delta(S)) = 1$ *or* $x(\delta(S)) = 2$;
3. $\chi(\delta(S))$ *is a linear combination of the characteristic vectors of its descendants in* $\mathcal{L}$ *(including possibly* $\chi(E(S))$*) and the characteristic vectors* $\chi(\delta(v))$ *of* $v \in S \cap T$;

Observe that special supernodes satisfy all the above properties. Intuitively, a special set has the same properties as a special supernode. The following lemma will complete the proof of Lemma 4.10, and hence Theorem 4.11.

Lemma 4.14 *For any rooted subtree of the forest $\mathcal{L} \neq \emptyset$ with the root S, we can distribute the tokens assigned to vertices inside S such that every vertex in $T \cap S$ and every node in the subtree gets at least two tokens and the root S gets at least three tokens. Moreover, the root S gets exactly three tokens only if S is a special set or $S = V$.*

Proof: First we prove some claims needed for the lemma.

Claim 4.15 *If $S \neq V$, then $|\delta(S)| \geq 2$.*

Proof: Since $S \neq V$, $x(\delta(S)) \geq 1$ is a valid inequality of the LP. As there is no 1-edge, $|\delta(S)| \geq 2$. □

Let the root be set S. We say a supernode C is a *member* of S if $C \subseteq S$ but $C \not\subseteq R$ for any child R of S. We also say a child R of S is a *member* of S. We call a member R of S *special*, if R is a special supernode (in which the supernode is a singleton vertex in T with degree three from Claim 4.12) or if R is a special set. In either case (whether the member is a supernode or set), a member has exactly one excess token only if the member is special. Special members also satisfy all the properties in Definition 4.13.

Recall that $E(S)$ denotes the set of edges with both endpoints in S. We denote by $D(S)$ the set of edges with endpoints in *different* members of S.

Claim 4.16 *If $S \in \mathcal{L}$ has r members then $x(D(S)) = r - 1$.*

Proof: For every member R of S, we have

$$x(E(R)) = |R| - |F(R)| - 1,$$

since either $R \in \mathcal{L}$, or R is a supernode in which case both LHS and RHS are zero. As $S \in \mathcal{L}$, we have

$$x(E(S)) = |S| - |F(S)| - 1$$

Now observe that every edge of $F(S)$ must be contained in $F(R)$ for some member R of S. Hence, we have the following, in which the sum is over R that are members of S.

$$x(D(S)) = x(E(S)) - \sum_R x(E(R))$$

$$= |S| - |S \cap F| - 1 - \sum_R (|R| - |F(R)| - 1)$$

$$= (|S| - \sum_R |R|) + \sum_R |F(R)| - |F(S)| + \sum_R 1 - 1$$

$$= (\sum_R 1) - 1 = r - 1$$

because $|S| = \sum_R |R|$ and $|F(S)| = \sum_R |F(R)|$. □

Claim 4.17 *Suppose a set $S \neq V$ contains exactly three special members R_1, R_2, R_3 and $|D(S)| \geq 3$. Then S is a special set.*

Proof: Note that $|\delta(S)| = |\delta(R_1)| + |\delta(R_2)| + |\delta(R_3)| - 2|D(S)| = 3 + 3 + 3 - 2|D(S)| = 9 - 2|D(S)|$. Since $S \neq V$, we have $|\delta(S)| \geq 2$ by Claim 4.15. As $|D(S)| \geq 3$, the only possibility is that $|D(S)| = 3$ and $|\delta(S)| = 3$, which satisfies the first property of a special set. Also, we have $x(\delta(S)) = x(\delta(R_1)) + x(\delta(R_2)) + x(\delta(R_3)) - 2x(D(S))$. As each term on the RHS is an integer, it follows that $x(\delta(S))$ is an integer. Moreover, as we do not have an 1-edge, $x(\delta(S)) < |\delta(S)| = 3$ and thus $x(\delta(S))$ is either equal to 1 or 2, and so the second property of a special set is satisfied. Finally, note that $\chi(\delta(S)) = \chi(\delta(R_1)) + \chi(\delta(R_2)) + \chi(\delta(R_3)) + \chi(E(R_1)) + \chi(E(R_2)) + \chi(E(R_3)) - 2\chi(E(S))$. Here, the vector $\chi(E(R_i))$ will be the zero vector if R_i is a special vertex. Since R_1, R_2, R_3

satisfy the third property of a special member, S satisfies the third property of a special set. □

The proof of Lemma 4.14 is by induction on the height of the subtree. In the base case, each member has at least one excess token and exactly one excess token when the member is special. Consider the following cases for the induction step.

1. S has at least four members. Each member has an excess of at least one. Therefore S can collect at least four tokens by taking one excess token from each.

2. S has exactly three members. If any member has at least two excess tokens, then S can collect four tokens, and we are done. Else each member has only one excess token and thus, by the induction hypothesis, is special. If $S = V$, then S can collect three tokens, and this is enough to force the contradiction to prove Lemma 4.10 since V is the root of the laminar family. Else, we have $x(D(S)) = 2$ from Claim 4.16. Because there is no 1-edge, we must have $|D(S)| > x(D(S)) = 2$. Now, it follows from Claim 4.17 that S is special and it only requires three tokens.

3. S contains exactly two members R_1, R_2. If both R_1, R_2 have at least two excess tokens, then S can collect four tokens, and we are done. Else, one of the members has exactly one excess token say R_1. Hence, R_1 is special by the induction hypothesis. We now show a contradiction to the independence of tight constraints defining x, and hence this case would not happen.

   Since S contains two members, Claim 4.16 implies $x(D(S)) = 1$. There is no 1-edge, therefore we have $|D(S)| = |\delta(R_1, R_2)| \geq 2$. Also, R_1 is special and thus $|\delta(R_1)| = 3$. We claim $\delta(R_1, R_2) = \delta(R_1)$. If not, then let $e = \delta(R_1) \setminus \delta(R_1, R_2)$. Then

$$x_e = x(\delta(R_1)) - x(\delta(R_1, R_2)) = x(\delta(R_1)) - x(D(S)).$$

   But $x(\delta(R_1))$ is an integer as R_1 is special and $x(D(S)) = 1$. Therefore, x_e is an integer which is a contradiction. Thus $\delta(R_1, R_2) = \delta(R_1)$. But then

$$\chi(E(S)) = \chi(E(R_1)) + \chi(\delta(R_1)) + \chi(E(R_2))$$

   if R_2 is a set (see Figure 4.7) or

$$\chi(E(S)) = \chi(E(R_1)) + \chi(\delta(R_1))$$

if R_2 is supernode.

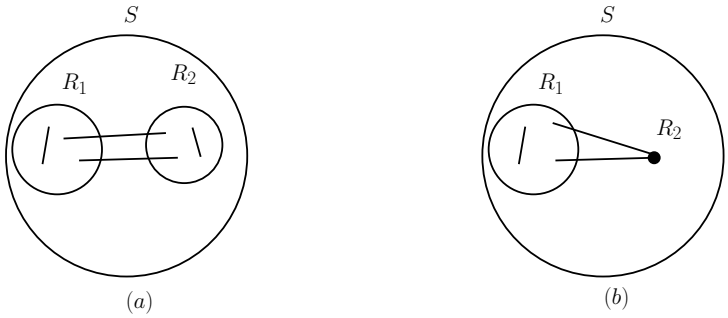


Figure 4.7: If $\delta(R_1, R_2) = \delta(R_1)$. In (a) we have R_2 is a set and in (b) R_2 is a supernode.

R_1 is special implies that $\chi(\delta(R_1))$ is a linear combination of the characteristic vectors of its descendants and the characteristic vectors $\{\chi(\delta(v))\colon\ v \in R_1 \cap T\}$. Hence, in either case $\chi(E(S))$ is spanned by $\chi(E(R))$ for $R \in \mathcal{L} \setminus \{S\}$ and $\chi(\delta(v))$ for $v \in S \cap T$ which is a contradiction to the inclusion of S in $\mathcal{L}$.

This completes the proof of Lemma 4.10, Lemma 4.14 and Theorem 4.11. □

5

Undirected Network Design with Degree Constraints

In this chapter we consider degree constrained general network design problems in undirected graphs and use iterative methods to achieve approximation algorithms. We prove the following results.

- In section 5.1 we give a polynomial time algorithm for the Minimum Bounded-Degree Steiner Network Design problem which returns a solution of cost at most twice the optimal and violates the degree bounds by at most an additive error of $6r_{max} + 3$, where r_{max} is the maximum connectivity requirement.
- In section 5.2, we consider the special case of Minimum Bounded-Degree Steiner Forest problem when the connectivity requirements are $\{0, 1\}$ for all pairs of vertices. We give a polynomial time algorithm which returns a solution with cost at most twice the optimal and violates the degree bounds by an additive error of at most three.

5.1 Minimum Bounded-Degree Steiner Network

Given connectivity requirements r_{uv} for all pairs of vertices, a *Steiner network* is a graph in which there are at least r_{uv} edge-disjoint paths between u and v for all pairs u, v. In the Minimum Bounded-Degree Steiner Network problem, we are given an undirected graph G with an edge cost for each edge, a connectivity requirement for each pair of vertices, and a degree upper bound B_v for each vertex v. The task is to find a minimum cost Steiner network H of G satisfying all the degree bounds, that is, $deg_H(v) \leq B_v$ for

all v. This problem captures many well-studied network design problems as special cases. For instance, a *Steiner forest* is a Steiner network with $r_{uv} \in \{0,1\}$ for all pairs; k-edge connected subgraph is a special case when $r_{uv} = k$ for all $u, v \in V$. Even the feasibility problem of finding a Steiner network satisfying all the degree bounds is already NP-hard since it generalizes the Hamiltonian path problem. In this section, we prove the following result.

Theorem 5.1 *There exists a polynomial time algorithm for the* Minimum Bounded-Degree Steiner Network *problem which returns a Steiner network H of cost at most* 2OPT *with degree violation at most $6r_{max} + 3$. Here* OPT *is the cost of an optimal solution which satisfies all the degree bounds, and $r_{max} := \max_{u,v}\{r_{uv}\}$.*

This result develops on the iterative rounding algorithm of Jain [53]. In Lau et al. [69], we first gave a $(2, 2B + 3)$-approximation algorithm using the iterative relaxation method. Here we achieve an *additive* violation in the degree bounds.

5.1.1 Linear Programming Relaxation

We begin by formulating a linear program for the problem. Set $f(S) = max_{u \in S, v \notin S}\ r_{uv}$ for each subset $S \subseteq V$. It is known that f is a *weakly supermodular* function [53], that is, for every two subsets X and Y, either

$$f(X) + f(Y) \leq f(X \cap Y) + f(X \cup Y)$$

or

$$f(X) + f(Y) \leq f(X - Y) + f(Y - X).$$

The following is a linear programming formulation for the Minimum Bounded-Degree Steiner Network problem, in which the degree constraints are on a subset of vertices $W \subseteq V$.

$$\begin{array}{lllll} \text{(LP-MBDSN)} & \text{minimize} & \displaystyle\sum_{e \in E} c_e\, x_e & & \\ & \text{subject to} \quad x(\delta(S)) & \geq & f(S) & \forall\, S \subseteq V \\ & \qquad\qquad\quad x(\delta(v)) & \leq & B_v & \forall\, v \in W \\ & \qquad\qquad\quad x_e & \geq & 0 & \forall\, e \in E \end{array}$$

When f is weakly supermodular function of the form discussed above, the above linear program can be efficiently separated and therefore, optimized over [53]. Also, if lower bounds on the degree are present then they can be incorporated with the connectivity constraints. This is achieved by setting $f(\{v\}) \leftarrow \max\{L_v, f(\{v\})$ where L_v is the lower bound on degree of vertex v. It is easy to verify that the updated function f remains weakly supermodular.

5.1.2 Characterization of Vertex Solutions

Let $\mathcal{F} = \{S \mid x(\delta(S)) = f(S)\}$ be the set of tight constraints from the connectivity requirement constraints. Recall that two sets X, Y are *intersecting* if $X \cap Y$, $X - Y$ and $Y - X$ are nonempty and that a family of sets is *laminar* if no two sets are intersecting. Since

$$x(\delta(X)) + x(\delta(Y)) \geq x(\delta(X \cap Y)) + x(\delta(X \cup Y)) \text{ and}$$

$$x(\delta(X)) + x(\delta(Y)) \geq x(\delta(X - Y)) + x(\delta(Y - X))$$

for any two subsets X and Y due to the cut function δ and f is weakly supermodular, it follows from standard uncrossing arguments (see e.g. [53]) that a vertex solution of the above linear program is characterized by a laminar family of tight constraints. This can be shown using an uncrossing argument as in Section 3.2.

Lemma 5.2 *Let the requirement function f of (LP-MBDSN) be weakly supermodular, and let x be a vertex solution of (LP-MBDSN) such that $0 < x_e < 1$ for all edges $e \in E$. Then, there exists a laminar family $\mathcal{L}$ of tight sets and a set $T \subseteq W$ with $x(\delta(v)) = B_v$ for each $v \in T$ such that:*

1. *The vectors $\chi(\delta(S))$ for $S \in \mathcal{L}$ and $\chi(\delta(v))$ for $v \in T$ are linearly independent.*
2. *$|E| = |\mathcal{L}| + |T|$.*
3. *For any set $S \in \mathcal{L}$, $\chi(\delta(S)) \neq \chi(\delta(v))$ for any $v \in W$.*

5.1.3 Iterative Algorithm

The iterative algorithm is given in Figure 5.1. In Step 2a we define a set of *high degree vertices* $W_h = \{v \in W | \sum_{e \in \delta(v)} x_e \geq 6 f_{max}\}$, where $f_{max} := \max_S f(S)$. Then in Step 2d we only pick an edge e with $x_e \geq \frac{1}{2}$ when both of its endpoints are not high degree vertices.

Minimum Bounded-Degree Steiner Network

1. Initialization $F \leftarrow \emptyset$, $f'(S) \leftarrow f(S)\ \forall S \subseteq V$.
2. While F is not a Steiner network
   (a) *Computing a vertex optimal solution:*
   Find a vertex optimal solution x satisfying f' and remove every edge e with $x_e = 0$. Set $W_h \leftarrow \{v \in W | \sum_{e \in \delta(v)} x_e \geq 6 f_{max}\}$ and $B_v \leftarrow \sum_{e \in \delta(v)} x_e$ for $v \in W$.
   (b) *Removing a degree constraint:*
   For every $v \in W$ with degree at most 4 in the support E, remove v from W.
   (c) *Picking an 1-edge:*
   For each edge $e = (u, v)$ with $x_e = 1$, add e to F, remove e from G, and decrease B_u, B_v by 1.
   (d) *Picking a heavy edge with both endpoints low:*
   For each edge $e = (u, v)$ with $x_e \geq 1/2$ and $u, v \notin W_h$, add e to F, remove e from G, and decrease B_u and B_v by 1/2.
   (e) *Updating the connectivity requirement function:*
   For every $S \subseteq V$: $f'(S) \leftarrow f(S) - |\delta_F(S)|$.
3. Return F.

Figure 5.1: Algorithm for the Minimum Bounded-Degree Steiner Network.

This is the key step to ensure that the degree bounds are only violated by an additive term and avoid the multiplicative factor of two on the degree bound improving on the previous iterative algorithm in Lau et al. [69].

First we show that the algorithm returns the solution with the claimed guarantees for cost and degree in Theorem 5.1 assuming that the algorithm always proceed to completion to return a feasible solution F. Then we show in Lemma 5.5 that for any vertex solution to the linear program one of the conditions must be satisfied.

Lemma 5.3 *If in each iteration one of the conditions in Step 2b, Step 2c or Step 2d is satisfied then the algorithm returns a Steiner network with cost at most twice the optimal linear programming solution to (LP-MBDSN) and degree bound of each vertex is violated by at most* $6r_{max} + 3$.

Proof: The proof is by a standard inductive argument. We give a short explanation. Note

that f' is a weakly supermodular function. Since we always pick an edge with $x_e \geq \frac{1}{2}$ and the remaining fractional solution is a feasible solution for the residual problem, the cost of the solution returned is at most twice the cost of the linear programming solution adapting the proof of Theorem 3.15 which we outline below. Let z^* denote the optimal solution to the initial linear program before the start of first iteration.

Claim 5.4 *In any iteration, if F denotes the current partial solution and x denotes the optimum solution to the residual linear program then*

$$c(F) + 2cx \leq 2cz^*.$$

Proof: The proof is by induction on the number of iterations. Initially, $F = \emptyset$ and $x = z^*$ and the claim holds. Now suppose the claim holds at a beginning of any iteration. If we remove a degree constraint in Step 2b then F does not change while the linear program is relaxed and hence its optimum can only decrease. Thus the term $c(F) + 2cx$ decreases at the end of iteration and the claim still holds true.

In the other case, we select an edge e with $x_e \geq \frac{1}{2}$ in Step (2c) or Step (2d). Let x_{res} denote the solution x, restricted to edges in $G \setminus e$, $F' = F \cup \{e\}$ and let x' denote the optimal solution to the residual linear program formed after this iteration . Since x_{res} is a feasible solution to this linear program we have $cx_{res} \leq cx'$. Thus we obtain that

$$c(F') + 2cx' \leq c(F) + c_e + 2cx_{res} \leq c(F) + 2cx \leq 2cz^*$$

where $c_e + 2cx_{res} \leq 2cx$ since $x_e \geq \frac{1}{2}$. Thus the induction hypothesis also holds in this case. □

For the guarantee on the degree bound, firstly observe that for any vertex v, we pick at most $B_v - 6f_{max}$ edges in Step 2c incident at v since the degree bound of v is reduced by one whenever such an edge is picked. In Step 2d, we pick at most $12f_{max} - 1$ edges incident at v since the degree bound is reduced by $\frac{1}{2}$ whenever we include such an edge and the degree constraint is removed before the bound reaches zero. Moreover, at most 4 edges can be picked incident at v once the degree constraint for v is removed. Hence, the number of edges picked which are incident at v is at most

$$B_v - 6f_{max} + 12f_{max} - 1 + 4 = B_v + 6f_{max} + 3,$$

as required. □

For the correctness of the algorithm, we shall prove the following key lemma in Section 5.1.4 which will ensure that the algorithm terminates with a feasible solution and complete the proof of Theorem 5.1.

Lemma 5.5 *Let x be a vertex solution of (LP-MBDSN), and W be the set of vertices with tight degree constraints, and $W_h = \{v \in W \mid \sum_{e \in \delta(v)} x_e \geq 6f_{max}\}$. Then at least one of the following holds.*

1. *There exists an edge e with $x_e = 1$.*
2. *There exists an edge $e = \{u, v\}$ with $x_e \geq 1/2$ and $u, v \notin W_h$.*
3. *There exists a vertex $v \in W$ such that $deg_E(v) \leq 4$.*

We say a vertex v is *owned* by a set S if $v \in S$ and S is the smallest set in $\mathcal{L}$ containing v.

5.1.4 A Counting Argument

We shall prove Lemma 5.5 by a counting argument. Suppose, by way of contradiction, that none of the conditions in the lemma holds. Then each edge e has $0 < x_e < 1$, and each edge e with $1 > x_e \geq 1/2$ (we call such an edge a *heavy edge*) must have at least one endpoint in W_h, and each vertex in W must have degree at least five. We give two tokens for each edge (the token assignment scheme is explained below) for a total of $2|E|$ tokens. Then, the tokens will be reassigned so that each member of $\mathcal{L}$ gets at least two tokens, each vertex in T gets at least two tokens and we still have some excess token left. This will contradict $|E| = |\mathcal{L}| + |T|$ of Lemma 5.2, and thus completes the proof.

The main difference from Jain's analysis is the existence of heavy edges (with an endpoint in W_h) which our algorithm is not allowed to pick. In the following, we say a vertex in W_h is a *high* vertex. Since there are heavy edges, a set $S \in \mathcal{L}$ may only have two edges in $\delta(S)$, and hence S may not be able to collect three tokens as in the proof of Jain [53]. To overcome this, we use a different token assignment scheme so that a similar induction hypothesis as Jain's would work.

Token assignment scheme: If $e = \{u, v\}$ is a heavy edge, $u \in W_h$ and $v \notin W$, then v

gets two tokens from e and u gets zero token. For every other edge e, one token is assigned to each endpoint of e.

Co-requirement: We also need to refine the definition of co-requirement from Jain [53] for the presence of heavy edges. Define

$$coreq(S) = \sum_{e \in \delta(S),\ x_e < 1/2} (1/2 - x_e) + \sum_{e \in \delta(S),\ x_e \geq 1/2} (1 - x_e).$$

It is useful to note that this definition reduces to Jain's definition if every edge e with $x_e \geq \frac{1}{2}$ is thought of as two parallel edges aiming to each achieves a value of $\frac{1}{2}$ and sharing the current x_e value equally (i.e. each gets $\frac{x_e}{2}$): summing $\frac{1}{2} - \frac{x_e}{2}$ over the two parallel edges gives $1 - x_e$, the second term.

After this initial assignment, each vertex in $V \setminus W_h$ receives at least as many tokens as their degree. Moreover, each vertex in $W \setminus W_h$ receive at least five tokens (as their degree is at least five). Note that a vertex $v \in W_h$ might not have any tokens if all the edges incident at it are heavy edges. By exploiting the fact that $f(S) \leq f_{max}$, however, we shall show that vertices in W_h can *get back* enough tokens during the inductive counting argument. Now we prove the following lemma which shows that the tokens can be reassigned as discussed previously.

Lemma 5.6 *For any subtree of $\mathcal{L}$ rooted at S, we can reassign tokens such that each vertex in $T \cap S$ gets at least two tokens, each set in the subtree gets at least two tokens, and the root S gets at least three tokens. Moreover, root S gets exactly three tokens only if $coreq(S) = \frac{1}{2}$.*

Proof: We now proceed by induction on the height of the subtree to prove Lemma 5.6. We first prove the base case of the induction hypothesis where we also show a crucial Claim 5.7, which handles all sets that own some vertices in W. We then use this claim in the main induction proof to complete the proof of Lemma 5.6.

Base Case of Lemma 5.6: S is a leaf node. First suppose that $S \cap W_h = \emptyset$. If there exists $v \in S \cap (W \setminus W_h)$, then v has at least five tokens. Since v only needs two tokens, it has three excess tokens which it can give to S. If there are two such vertices or S owns another endpoint, then S gets at least four tokens as required. Otherwise, we have $\chi(\delta(v)) = \chi(\delta(S))$ which is a contradiction to the linear independence of characteristic vectors in Lemma 5.2. Hence, we assume $S \cap W = \emptyset$. Then S can get at least $\delta(S)$ tokens from the vertices owned by S. Note that $|\delta(S)| \geq 2$, as $x(\delta(S))$ is an integer and there is

no 1-edge. If $|\delta(S)| \geq 4$, then S gets four tokens. If $|\delta(S)| = 3$ and $|\delta(S)|$ contains a heavy edge, then S can get four tokens from the vertices it owns, since an endpoint $v \notin W$ of a heavy edge has 2 tokens by the token assignment scheme. If it does not contain a heavy edge, then S receives three tokens and $coreq(S) = \frac{1}{2}$. If $|\delta(S)| = 2$, then at least one edge is a heavy edge. If both edges are heavy then S can get four tokens, else if only one edge is heavy then it gets three tokens and $coreq(S) = \frac{1}{2}$.

We now consider the case that S owns a vertex in W_h, and show that S can collect enough tokens for the inductive argument. The following claim is the key to deal with degree constraints, which uses crucially the parameter f_{max}. This claim holds even when S is not a leaf in the laminar family, and will also be used in the induction step.

Claim 5.7 *Suppose S owns $r \geq 1$ vertices in W_h. Then the number of excess tokens from the children of S, plus the number of tokens owned by S, plus the number of tokens left with vertices in W_h owned by S is at least $2r + 4$.*

Proof: Let S have c children. As each child has at least one excess token by the induction hypothesis, if $c \geq 6r$ then we have $6r$ tokens which is at least $2r + 4$. Hence, we assume that $c < 6r$.

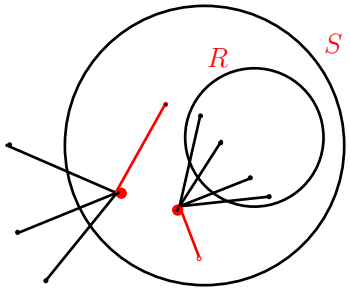


Figure 5.2: In this example, red vertices are in W_h which donate tokens for edges incident at them to the other endpoint. Observe that the tokens for black edges are still available for S which the tokens for black edges are not.

Let $B := \sum_v B_v \geq \sum_v 6 f_{max} = 6r f_{max}$, where the sum is over all vertices $v \in W_h$ owned by S. Intuitively, vertices in W_h owned by S would have collected a total of B tokens if the two tokens at each edge is distributed evenly. But by the token assignment scheme, vertices in W_h owned by S may not get any token for heavy edges incident on them. We are going to show that these vertices can still "get back" the two tokens they need for the inductive argument (see Figure 5.2).

For a child R of S, as $x(\delta(R)) = f(R) \leq f_{max}$, at most f_{max} units of B come from the edges in $\delta(R)$. Similarly, at most f_{max} units of B come from the edges in $\delta(S)$. Hence, there are at least $f_{max}(6r - c - 1)$ units of B coming from the edges with both endpoints owned by S. Since there is no 1-edge, there are at least $f_{max}(6r-c-1)+1$ such endpoints from those edges. Let $e = \{u, v\}$ be such an edge with $v \in W_h$ owned by S. If $u \in W$, then both u and v get one token from e in the initial assignment. If $u \notin W$, then u gets two tokens from e in the initial assignment, but these two tokens are owned by S. So, the number of tokens owned by S plus the number of tokens left with vertices in W_h owned by S is at least $f_{max}(6r - c - 1) + 1$. Furthermore, S can also collect one excess token from each child. So, the total number of tokens S can collect is at least $f_{max}(6r-c-1)+c+1$, which is a decreasing function of c. As $c < 6r$, the number of tokens is minimized at $c = 6r - 1$, which is at least $6r \geq 2r + 4$. □

In the base case when S owns a vertex in W_h, using Claim 5.7, S can collect $2r + 4$ tokens. So these tokens can be redistributed so that S has 4 tokens and each vertex in W_h owned by S has 2 tokens, which is enough for the induction hypothesis.

Induction Step: The presence of heavy edges with $x_e \geq \frac{1}{2}$ introduces some difficulties in carrying out the inductive argument in [53]. We need to prove some lemmas which work with the new notion of co-requirement and the presence of heavy edges.

For any set S, let $wdeg(\delta(S))$

$$= |\{e \in \delta(S) : 0 < x_e < \frac{1}{2}\}| + 2|\{e \in \delta(S) : x_e \geq \frac{1}{2}\}|$$

be the *weighted degree* of S. This definition is keeping with the idea that each edge with $x_e \geq \frac{1}{2}$ is thought of as two parallel edges. Observe that for any $v \notin W$, it receives exactly $wdeg(v)$ tokens in the initial assignment as it gets one token for each edge and two tokens for all heavy edges incident at it. S can take all the tokens for all the vertices it owns which are not in W. We call these the *tokens owned by* S. Let $G' = (V, E')$ be the graph formed by replacing each heavy edge e by two edges e' and e'' such that $x_{e'} = x_{e''} = \frac{x_e}{2}$. Observe that

$$coreq(S) = \sum_{e \in \delta(S),\ x_e < 1/2} (1/2 - x_e) + \sum_{e \in \delta(S),\ x_e \geq 1/2} (1 - x_e)$$

$$= \sum_{e \in \delta(S) \cap E'} (1/2 - x_e),$$

and $wdeg(\delta(S)) = |\delta'(S)|$ where $\delta'(S) = \{e \in E' : e \in \delta(S)\}$. Observe that $coreq(S)$ is integral or *semi-integral* (half-integral but not integral) depending on whether $\delta'(S)$ is even or odd. We first prove the same technical lemma as in [53] with the new definitions of co-requirements and weighted degrees.

Claim 5.8 *Let S be a set in $\mathcal{L}$ which owns α tokens and has β children where $\alpha+\beta = 3$ and does not own any vertex of W. Furthermore, each child of S, if any, has a co-requirement of $\frac{1}{2}$. Then the co-requirement of S is $\frac{1}{2}$.*

Proof: Since each child R of S has a co-requirement of half, this implies that $|\delta'(R)|$ is odd. Note that we assume S does not own any vertex of W. Using these facts and that $\alpha + \beta = 3$, a simple case analysis (as in Exercise 23.3 of [99]) can be used to show that $|\delta'(S)|$ is odd. Hence, the co-requirement of S is semi-integral. Now, we show that $coreq(S) < \frac{3}{2}$ proving the claim. Clearly,

$$coreq(S) = \sum_{e \in \delta'(S)} (1/2 - x_e) \leq \sum_{R} coreq(R) + \sum_{e} (1/2 - x_e),$$

where the first sum is over all children R of S and second sum is over all edges for which S owns a token. Since $\alpha+\beta = 3$, there are a total of three terms in the sum. Since, any term in the first sum is $\frac{1}{2}$ and in the second sum is strictly less than $\frac{1}{2}$, if $\alpha > 0$, we then have $coreq(S) < \frac{3}{2}$ which proves the claim. So, assume $\alpha = 0$, i.e. S does not own any tokens. In this case, edges incident to children of S cannot all be incident at S since otherwise it will violate the linear independence of characteristic vectors in $\mathcal{L}$ in Lemma 5.2, and therefore we have $coreq(S) < \sum_R coreq(R) = \frac{3}{2}$ proving the claim. □

We are now ready to prove that the induction step holds, in which S has at least one child. If S owns a vertex in W_h then Claim 5.7 shows that the induction hypothesis holds. Henceforth, we assume that S does not own any vertices of W_h. Suppose S owns some vertices in $W \setminus W_h$. Each such vertex gets at least five tokens. It needs only two tokens and hence can give three excess tokens to S. As S has at least one child R, R can give at least one excess token to S, and hence S gets at least four tokens as required.

For the rest of the cases, we assume that S does not own any vertex of W, and hence the remaining case analysis is very similar to that of Jain, with a different definition of co-requirement.

- S has at least four children. Then S can take one excess token from each child.

- S has exactly three children. If any child S has two excess tokens or if S owns a vertex then S can get four tokens. Else, each of the three children of S has a co-requirement of half and S owns no vertices. Then, by Claim 5.8, we have that S has co-requirement of $\frac{1}{2}$ and it only needs three tokens.

- S has exactly two children R_1 and R_2. If both of them have two excess tokens then we are done. Else, let R_1 have exactly one token and hence it has co-requirement of $\frac{1}{2}$ by the induction hypothesis. We now claim that S owns an endpoint. For the sake of contradiction suppose S does not own any endpoint. Then, if there are α edge between R_1 and R_2 in E' (where we replace each heavy edge by two parallel edges), we have
$$|\delta'(S)| = |\delta'(R_1)| + |\delta'(R_2)| - 2\alpha$$
As R_1 has a co-requirement of half, we have $|\delta'(R_1)|$ is odd and hence $\delta'(S)$ and $\delta'(R_2)$ have different parity and hence different co-requirements. The co-requirements of S and R_2 can differ by at most the co-requirement of R_1 which is exactly half. Since, $\chi(\delta'(S)) \neq \chi(\delta'(R_1)) + \chi(\delta'(R_2))$, there must be an edge between R_1 and R_2 and therefore, $coreq(S) < coreq(R_2) + \frac{1}{2}$. Similarly, $\chi(\delta'(R_2)) \neq \chi(\delta'(S)) + \chi(\delta'(R_1))$ and therefore there is an edge in $\delta'(S) \cap \delta'(R_1)$ which implies that $coreq(R_2) < coreq(S) + \frac{1}{2}$. Thus, their co-requirements are equal which is a contradiction. Thus S owns at least one endpoint.

  If S owns at least two endpoints or R_2 has two excess tokens, then we have four tokens for S. Otherwise, by Claim 5.8, we have that co-requirement of S is half and it needs only three tokens.

- S has exactly one child R. Since both sets S and R are tight we have that $x(\delta(S)) = f'(S)$ and $x(\delta(R)) = f'(R)$. Since $\chi(\delta(S))$ and $\chi(\delta(R))$ are linearly independent, subtracting the two equations we have that $x(\delta(S)\Delta\delta(R))$ (Δ denotes symmetric difference) is an positive integer. Also, there are no 1-edges present and so $|\delta(S)\Delta\delta(R)| \geq 2$, and each edge in the symmetric difference gives one token to S. Thus S owns at least two endpoints. If S owns three endpoints or R has two excess tokens then S can get four tokens. Otherwise, S has exactly two endpoints and exactly one child which has co-requirement of $\frac{1}{2}$. Then by Claim 5.8, S has a co-requirement of $\frac{1}{2}$.

This completes the proof of Lemma 5.6, which assigns two tokens to each set in the laminar family $\mathcal{L}$ and each vertex in T which is contained in some set $S \in \mathcal{L}$. For vertices in T which are not contained in any set $S \in \mathcal{L}$ we also have enough tokens. Observe that

each vertex $v \in W \setminus W_h$ receives at least five tokens. For vertices in W_h not contained in any set $S \in \mathcal{L}$, an argument identical to Claim 5.7 with $S = V$ will give at least two tokens to each vertex in W_h.

Thus we have that $2|E| > 2|\mathcal{L}| + 2|T|$, which contradicts Lemma 5.2. Therefore, one of the conditions in Lemma 5.5 holds, and hence we have Theorem 5.1. □

Integrality Gap Example. In Figure 5.3 we show that the linear program (LP-MBDSN) has an integrality gap of $B + \Omega(r_{max})$ and therefore Theorem 5.1 is nearly tight.

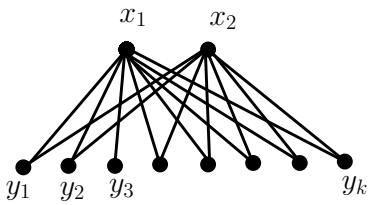


Figure 5.3: In this example, we have a complete bipartite graph $B = (X, Y, E)$ where $X = \{x_1, x_2\}$ and $Y = \{y_1, \ldots, y_k\}$. We set the connectivity requirements between y_i and y_j to be 1 for all i, j, between x_1 and x_2 to be $\frac{k}{2}$, and 0 otherwise. The fractional solution where all edges have fractional value $\frac{1}{2}$ is the optimal solution, in which the degree of x_1 and x_2 is equal to $\frac{k}{2} = \Delta_f^*$. On the other hand, it can be seen that in any integer solution, the degree of x_1 and x_2 must be at least $\frac{3}{4}k = \frac{3}{2}\Delta_f^*$. This example also shows that the any feasible solution must cost twice the cost of optimal LP solution and must violate the degree bounds by at least $\frac{r_{max}}{2}$ for some vertex v.

5.2 Minimum Bounded-Degree Steiner Forest

In this section we study the Minimum Bounded-Degree Steiner Forest problem which is a special case Minimum Bounded-Degree Steiner Network problem with $r_{uv} \in \{0, 1\}$ for each pair of vertices $u, v \in V$. We show an improved analysis in this particular case and prove the following theorem.

Theorem 5.9 *There exists a polynomial time algorithm for the* Minimum Bounded-Degree Steiner Forest *problem which returns a Steiner forest F of cost at most* 2OPT *with degree violation at most 3 (i.e. $deg_F(v) \leq B_v + 3 \ \forall v \in F$). Here* OPT *is the cost of an optimal solution which satisfies all the degree bounds.*

The linear program is identical to the linear program in Section 5.1 for the Minimum Bounded-Degree Steiner Network problem with the extra restriction that f is a

$\{0,1\}$-valued function. Moreover, the same characterization of vertex solutions follows as in Lemma 5.2 which we will use in the next section.

5.2.1 Improved Iterative Algorithm

Our algorithm is an iterative relaxation algorithm as shown in Figure 5.4. We pick a heavy edge ($1 > x_e \geq \frac{1}{2}$) only if both endpoints do not have degree constraints. Also, by only picking edges with no degree constraints, there is no need to update the degree bounds fractionally. Note also that the relaxation step has been generalized to remove a degree constraint when a vertex has degree at most $B_v + 3$.

Minimum Bounded-Degree Steiner Forest

1. Initialization $F \leftarrow \emptyset$, $f'(S) \leftarrow f(S)\ \forall S \subseteq V$.
2. While F is not a Steiner forest
   (a) *Computing a vertex optimal solution:*
   Find a vertex optimal solution x satisfying f' and remove every edge e with $x_e = 0$.
   (b) *Removing a degree constraint:*
   For every $v \in W$ with degree at most $B_v + 3$ in the current support graph, remove v from W.
   (c) *Picking an 1-edge:*
   For each edge $e = \{u, v\}$ with $x_e = 1$, add e to F, remove e from G, and decrease B_u, B_v by 1.
   (d) *Picking a heavy edge with no degree constraints:*
   For each edge $e = \{u, v\}$ with $x_e \geq \frac{1}{2}$ and $u, v \notin W$, add e to F and remove e from G.
   (e) *Updating the connectivity requirements:*
   Set $f'(S) \leftarrow f(S) - \delta_F(S)$.
3. Return F.

Figure 5.4: Algorithm for Minimum Bounded-Degree Steiner Forest.

The following lemma shows that the algorithm will always terminate successfully.

Lemma 5.10 *Every vertex solution x of (LP) must satisfy one of the following:*

1. *There is an edge e with $x_e = 0$ or $x_e = 1$.*

2. *There is an edge* $e = \{u, v\}$ *with* $x_e \geq \frac{1}{2}$ *and* $u, v \notin W$.

3. *There is a vertex* $v \in W$ *with* $deg(v) \leq B_v + 3$.

Note that the updated connectivity requirement function f' is also a weakly supermodular function. With Lemma 5.10, using a simple inductive argument as in the previous section, it can be shown that the algorithm returns a Steiner forest of cost at most twice the optimal cost and the degree of each vertex is at most $B_v + 3$. The rest of this section is devoted to the proof of Lemma 5.10.

5.2.2 A Refined Counting Argument

The proof of Lemma 5.10 is by contradiction. Let $\mathcal{L}$ be the laminar family and $T \subseteq W$ be the set of tight vertices defining the vertex optimal solution x as in Lemma 5.2. The contradiction is obtained by a counting argument. Each edge in E is assigned two tokens. Then the tokens will be redistributed such that each member of $\mathcal{L}$ and each vertex in T get at least two tokens, and there are still some extra tokens left. This will give us a contradiction to Lemma 5.2 that $|E| = |\mathcal{L}| + |T|$.

As before we say an edge is *heavy* if $x_e \geq \frac{1}{2}$. If all conditions of Lemma 5.10 do not hold, we must have that there is no 0-edge and no 1-edge, every heavy edge has an endpoint in W, and each vertex $v \in W$ has at least $B_v + 4$ edges incident at it.

Token assignment scheme: The two tokens for an edge $e = \{u, v\}$ are assigned by the following rules.

1. One token of e is assigned to u and the other token of e is assigned to v.

2. If $e = (u, v)$ is a heavy edge with $v \in W$ and u is not contained in the smallest set in $\mathcal{L}$ containing v, then the token of e for v is reassigned to the smallest set $S \in \mathcal{L}$ containing both u and v (see Figure 5.5).

Classes: Let R be a set in $\mathcal{L}$. An edge $e = \{u, v\}$ is an *out-heavy edge* of R if $u \in R \setminus W$ and $v \in W \setminus R$ and $x_e \geq \frac{1}{2}$. The following definition is important to the analysis. For a set $R \in \mathcal{L}$, we define R to be of

- *Class Ia*: if $|\delta(R)| = 2$ and R has one out-heavy edge e with $x_e > \frac{1}{2}$.
- *Class Ib*: if R has two out-heavy edges. (Note that $|\delta(R)| = 2$ in this case and each has value $\frac{1}{2}$).

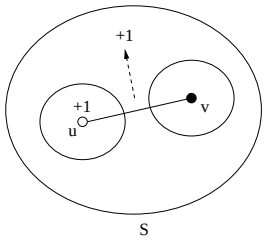


Figure 5.5: Rule (2) of the token assignment scheme. This is a new rule which is useful in collecting an extra token for S. Here, the degree constraint for vertex u has been removed but the degree constraint for vertex v is present.

- *Class IIa*: if $|\delta(R)| = 3$ and $x_e < \frac{1}{2}$ for each edge $e \in \delta(R)$.
- *Class IIb*: if R has one out-heavy edge.
- *Class III*: otherwise.

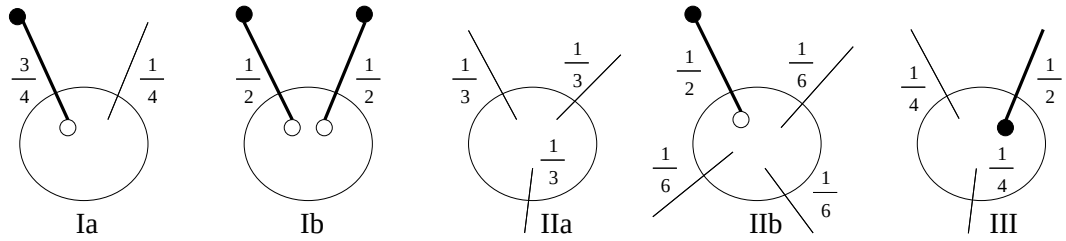


Figure 5.6: The figure shows examples of sets of each class. A vertex without degree constraint is white, otherwise it is black. (An endpoint without a vertex shown means that this information is not important.) A heavy edge is represented by a thick line. Note the definition of Class Ia, Class Ib and Class II require out-heavy edges. The rightmost example is a Class III set, although it has a heavy edge.

The following lemma shows that the tokens can be redistributed so that each member of $\mathcal{L}$ and each vertex in W gets at least two tokens. The proof is by induction on the laminar family.

Lemma 5.11 *For any subtree of the laminar family $\mathcal{L}$ rooted at S, we can redistribute tokens in S such that*

1. *Every vertex in $T \cap S$ gets at least two tokens.*
2. *Class I sets in the subtree get at least two tokens.*

3. *Class II sets in the subtree get at least three tokens.*

4. *Class III sets in the subtree get at least four tokens.*

Proof: Here is a brief outline of the proof. First we show in Claim 5.12 that a set owning at least two vertices in W can collect enough tokens; this uses the fact that f is a 0-1 function. Then a series of claims, Claim 5.13, Claim 5.14, Claim 5.15 and Claim 5.16 are used to show that a set owning exactly one vertex in W can collect enough tokens. Then the remaining cases consider sets which do not own any vertex in W, which rely crucially on Claim 5.17. We remark that Rule (2) of the token assignment scheme and the asymmetry in the definition of out-heavy edges are used in Claim 5.17. Now we start the proof by proving Claim 5.12.

Claim 5.12 *If $S \in \mathcal{L}$ owns two or more vertices in W then the induction claim holds.*

Proof: Suppose S owns $w_1, \ldots, w_r \in W$. Since $B_v \geq 1$ for all $v \in W$, by Step 2b of the algorithm, each vertex w_i is of degree at least 5. Since $f(S) = 1$, $\delta(S)$ can have at most two heavy edges. Hence, by the token assignment scheme, there are at least $5r - 2$ tokens assigned to $w_1, \ldots, w_r$ which have not be reassigned by Rule (2). Since each vertex in $W \cap S$ needs only two tokens, there are still $3r - 2$ extra tokens left. If $r \geq 2$, then S can collect at least 4 tokens, as required. □

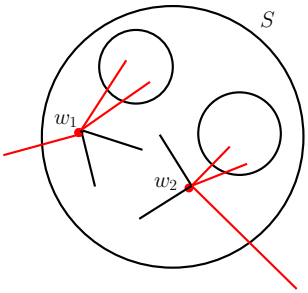


Figure 5.7: Here w_1 and w_2 are vertices in W owned by S. The red edges are heavy edges and there can be at most two such edges in $\delta(T)$ for any tight set T (here S and its children). Observe that vertices in W still retain tokens for $2 * 3 - 2 = 4$ tokens (for the black edges).

Hence suppose w is the only vertex in W owned by S.

Claim 5.13 *Let S be the set that owns $w \in W$. Then w is assigned at least four tokens, and is assigned exactly four tokens only if:*

1. *$deg(w) = 5$, $B_w = 1$, and there is one heavy edge of $\delta(S)$ in $\delta(w)$.*

2. *$deg(w) = 6$, $B_w = 2$, and there are two heavy edges of $\delta(S)$ in $\delta(w)$. In this case, $\delta(S) = \delta(S) \cap \delta(w)$.*

Proof: Since $f(S) = 1$, $\delta(S)$ can have at most two heavy edges. So w receives one token for each edge incident at w except for the heavy edges in $\delta(w) \cap \delta(S)$. By Step 2b of the algorithm, w is of degree at least five. Hence, if there is no heavy edge in $\delta(S) \cap \delta(w)$, then w receives five tokens. Suppose that $\delta(S) \cap \delta(w)$ has only one heavy edge. Thus w receives $deg(w) - 1 \geq 4$ tokens and exactly four tokens only if $deg(w) = 5$ and $B_w = 1$.

Suppose $\delta(w) \cap \delta(S)$ has two heavy edges, then $B_w = x(\delta(w)) \geq 2$ since there are no 0-edges. Therefore, $deg(w) \geq 6$ by Step 2b of the algorithm. Thus, w receives at least $deg(w) - 2 \geq 4$ tokens and exactly four tokens only if $deg(w) = 6$, $B_w = 2$ and $\delta(S) = \delta(w) \cap \delta(S)$ contains two heavy edges. □

We now show via a series of claims that when S owns exactly one vertex in W, there are enough tokens for S and the vertex $w \in W$ it owns.

Claim 5.14 *If S owns one vertex in W and has one Class III child or two Class II children then there are enough tokens for w and S.*

Proof: Let w be the vertex in W that S owns. From Claim 5.13, it follows that w receives at least four tokens and therefore it has two extra tokens. Each Class III child also receives two extra tokens by the induction hypothesis and each Class II child receives one extra token. Hence, S can collect two tokens from w and two tokens from its Class III child or the two class II children whichever the case. □

Now, we show that induction hypothesis holds even when S owns exactly one Class II child.

Claim 5.15 *If S owns one vertex in W and has no Class III child and exactly one Class II child then there are enough tokens for w and S.*

Proof: Following the argument in Claim 5.14, S can receive at least two extra tokens from w and one extra token from its class II child, say R_1. Let $R_2, \ldots, R_l$ be the class I children of S where $l \geq 1$ ($l = 1$ implies that there are no class I children) . If w has another extra

token or S owns an endpoint or $\delta(S)$ has an out-heavy edge then the claim holds. Thus any out-heavy edge incident in $\delta(R_i)$ is in $\delta(w)$ or is an in-heavy edge in $\delta(R_1)$. In the latter case R_1 is of Class IIb. Moreover, w satisfies one of the two cases in Claim 5.13. We now consider the two cases depending on which of the conditions hold.

1. $deg(w) = 5$, $B_w = 1$, and there is one heavy edge of $\delta(S)$ in $\delta(w)$. Thus, w cannot have another heavy edge incident at it. If R_1 is of Class IIb then the out-heavy edge in $\delta(R_i)$ must also be in $\delta(S)$ which contradicts that S needs four tokens. Thus R_1 is of Class IIa and also S does not have any Class I children. But we have

$$\begin{aligned} x(\delta(R_1)) &= x(\delta(R_1) \cap \delta(w)) + x(\delta(R_1) \cap \delta(S)) = 1 \\ x(\delta(S)) &= x(\delta(S) \cap \delta(R_1)) + x(\delta(S) \cap \delta(w)) = 1 \\ x(\delta(w)) &= x(\delta(w) \cap \delta(R_1)) + x(\delta(w) \cap \delta(S)) = 1 \\ \implies x(\delta(w) \cap \delta(R_1)) &= x(\delta(w) \cap \delta(S)) = x(\delta(R_1) \cap \delta(S)) = \frac{1}{2} \end{aligned}$$

   But $|\delta(R_1)| = 3$ thus either $|\delta(R_1) \cap \delta(w)| = 1$ or $|\delta(R_1) \cap \delta(S)| = 1$. In either case, there is a heavy edge in $\delta(R_1)$ contradicting that R_1 is of Class IIa.

2. $deg(w) = 6$, $B_w = 2$, and there are two heavy edges of $\delta(S)$ in $\delta(w)$. In this case, $\delta(S) = \delta(S) \cap \delta(w)$. Thus w can have at most one more heavy edge incident at it. Thus S has at most one class I child R_2. Therefore we have

$$\begin{aligned} x(\delta(R_1)) &= x(\delta(R_1) \cap \delta(w)) + x(\delta(R_1) \cap \delta(R_2)) = 1 \\ x(\delta(S)) &= x(\delta(S) \cap \delta(w)) = 1 \\ x(\delta(R_2)) &= x(\delta(w) \cap \delta(R_2)) + x(\delta(w) \cap \delta(R_2)) = 1 \\ x(\delta(w)) &= x(\delta(w) \cap \delta(S)) + x(\delta(w) \cap \delta(R_1)) + x(\delta(w) \cap \delta(R_2)) = 2 \\ \implies x(\delta(w) \cap \delta(R_1)) &= x(\delta(w) \cap \delta(R_2)) = x(\delta(R_1) \cap \delta(R_2)) = \frac{1}{2} \end{aligned}$$

   Thus again there is a heavy edge in $\delta(R_1)$ and R_1 must be of Class IIb. Also R_2 is of class Ib, since there is no edge e in $\delta(R_2)$ with $x_e > \frac{1}{2}$. Thus, there is a single heavy edge between R_1 and R_2. But this edge gives one token to S by Rule 2 giving four tokens to S.

This completes the proof of Claim 5.15 □

We now assume that S only has class I children and prove that S can collect enough tokens.

Claim 5.16 *If S has only class I children and own one vertex w in W, then S can collect enough tokens.*

Proof: If S is of class I then it only needs two tokens which it can take from w. Similarly, if S is of Class IIa then w must have three extra tokens since there is no heavy edge in $\delta(S)$ and therefore the two condition in Claim 5.13 cannot hold. Thus S is of either Class IIb or Class III. Let $R_1, \ldots, R_l$ be the Class I children of S. There is no heavy edge with one endpoint in one Class I child and another endpoint in another Class I child, by the definition of Class I child. Let h be the number of out-heavy edges of $R_1 \cup \ldots \cup R_l$ that are also in $\delta(S)$. The goal now is to collect two tokens for w and $4-h$ tokens for S. Observe that $h \leq 1$ since S is of Class IIb or Class III.

1. Suppose $h = 1$. Let e_1 be an out-heavy edge in $\delta(S) \cap \delta(R_1)$. Then S is of Class IIb, and only needs three tokens. If w has at least five tokens, then there are enough tokens. So assume w has exactly four tokens. Since $f(S) = 1$ and there is already a heavy edge $e_1 \in \delta(S)$, Case (2) of Claim 5.13 cannot happen. So, by Claim 5.13, the only possibility is that $B_w = 1$, $deg(w) = 5$ and there is one heavy edge f in $\delta(w) \cap \delta(S)$. Hence $\delta(S) = \{e_1, f\}$. Since $B_w = 1$, w does not have any other heavy edges incident at it. Therefore S has only one child since the out-heavy edge of another class I child, say R_2, must be incident at $\delta(w)$. Since $|\delta(w)| = 5$, $|\delta(R_1)| = 2$, and $|\delta(w) \cap \delta(S)| = 1$, there must be an edge $\{w, x\}$ with $x \in S - R_1$ for which S owns a token as required.

2. Suppose $h = 0$. Then every out-heavy edge of R_i is incident on w and S is of Class III. If $deg(w) \geq 8$, then w has at least six tokens, and it can give four tokens to S. Else we have the the following cases.

   (a) Suppose $deg(w) = 7$. Then by Step 2b of the algorithm, $B_w \leq 3$. If there is at most one heavy edge in $\delta(w) \cap \delta(S)$, then w loses only one token and has at least six tokens by the token assignment scheme, and this is enough. So assume that there are two heavy edges in $\delta(w) \cap \delta(S)$, and hence $|\delta(w) \cap \delta(S)| = |\delta(S)| = 2$. If S owns an endpoint, then S can collect one more token, and this is enough. So further assume that S does not own an endpoint. Therefore $\delta(w) \setminus \delta(S) = \delta(w, \cup_{i=1}^{l} R_i)$. We shall prove that this would not happen. Note

that $|\delta(w, \cup_{i=1}^{l} R_i)| = |\delta(w) \setminus \delta(S)| = |\delta(w)| - |\delta(w) \cap \delta(S)| = 7 - 2 = 5$. Since $\delta(R_i) = 2$ for each Class I child, this implies that $l \geq 3$. Each R_i has an out-heavy edge e_i incident on w. Suppose $l \geq 4$. Since $deg(w) = 7$, this implies that $B_w = x(\delta(w)) > x(\delta(w) \cap \delta(S)) + x_{e_1} + x_{e_2} + x_{e_3} + x_{e_4} \geq 3$, a contradiction. So S must have exactly three children R_1, R_2, R_3. Since $|\delta(w, \cup_{i=1}^{l} R_i)| = 5$, there are exactly two children with $|\delta(R_i)| = |\delta(w, R_i)| = 2$, say R_1 and R_2. But then $B_w = x(\delta(w)) = x(\delta(w) \cap \delta(S)) + x(\delta(R_1)) + x(\delta(R_2)) + x_{e_3} \geq 1 + 1 + 1 + \frac{1}{2} > 3$, a contradiction.

(b) Suppose $deg(w) = 6$. Then by Step 2b of the algorithm, $B_w \leq 2$. If there is no heavy edge in $\delta(w) \cap \delta(S)$, then w has at least six tokens, and this is enough. So assume that there is at least one heavy edge in $\delta(w) \cap \delta(S)$.

Suppose there are two heavy edges in $\delta(w) \cap \delta(S)$, and hence $|\delta(w) \cap \delta(S)| = |\delta(S)| = 2$. If there are two edges in $\delta(w)$ with the other endpoint in $S - \cup_{i=1}^{l} R_i$, then S can collect two more tokens, as required. Otherwise, since $deg(w) = 6$ and $\delta(R_i) = 2$, S must have at least two children R_1 and R_2. Each R_i has a heavy edge e_i incident at w. So, since $deg(w) = 6$, this implies $B_w = x(\delta(w)) > x(\delta(w) \cap \delta(S)) + x_{e_1} + x_{e_2} \geq 2$, a contradiction.

Henceforth, we assume that there is exactly one heavy edge f in $\delta(w) \cap \delta(S)$. So, w has at least five tokens, and we need to collect one more token. Each R_i has a heavy edge e_i incident at w. If S has at least three children, then $B_w > x_f + x_{e_1} + x_{e_2} + x_{e_3} \geq 2$, a contradiction. So S has at most two children. On the other hand, since S has at least one child, and so $B_w > x_f + x_{e_1} \geq 1$ and hence $B_w = 2$. If S owns an endpoint, then S can collect one more token, as required. So further assume that S does not own an endpoint.

Suppose S has exactly two children R_1 and R_2. If $\delta(w, R_1) = \delta(R_1)$, then $B_w > x_f + x(\delta(R_1)) + x_{e_2} \geq 2$, a contradiction. Hence $\delta(w, R_1) = \{e_1\}$ and $\delta(w, R_2) = \{e_2\}$. Suppose $\delta(R_1, R_2) \neq \emptyset$. Let $\delta(R_1, R_2) = \{e\}$. Then $x_e + x_{e_1} = x(\delta(R_1)) = f(R_1) = 1$, and $x_e + x_{e_2} = x(\delta(R_2)) = f(R_2) = 1$, and $x_{e_1} + x_{e_2} = x(\delta(w)) - x(\delta(w) \cap \delta(S)) = B_w - f(S) = 1$. Therefore, $x_e = x_{e_1} = x_{e_2} = \frac{1}{2}$. But then e is a heavy edge between two Class I children, a contradiction. So $\delta(R_1, R_2) = \emptyset$. Let $R_1 = \{e_1, f_1\}$ and $R_2 = \{e_2, f_2\}$. The only possibility left is $f_1 \in \delta(S)$ and $f_2 \in \delta(S)$. Note that $x_{e_1} + x_{f_1} = x(\delta(R_1)) = f(R_1) = 1$, and $x_{e_2} + x_{f_2} = x(\delta(R_2)) = f(R_2) = 1$, and $x_{e_1} + x_{e_2} + x(\delta(w) \cap \delta(S)) = x(\delta(w)) = B_w = 2$, and $x_{f_1} + x_{f_2} + x(\delta(w) \cap \delta(S)) = x(\delta(S)) = f(S) = 1$. Hence $x(\delta(w) \cap \delta(S)) = \frac{1}{2}$. Since $f \in \delta(w) \cap \delta(S)$ is a heavy edge, this implies

that f is the only edge in $\delta(w) \cap \delta(S)$. But then $\delta(w) = \{e_1, e_2, f\}$ and hence $deg(w) = 3$, a contradiction.

Henceforth assume that S has exactly one child R_1. If S owns an endpoint, then S can collect one more token, as required. So further assume that S does not own an endpoint. We prove that this case would not happen. Then $x(\delta(w) \cap \delta(S)) + x(\delta(w, R_1)) = x(\delta(w)) = B_w = 2$, and $x(\delta(w, R_1)) + x(\delta(R_1) \cap \delta(S)) = x(\delta(R_1)) = f(R_1) = 1$, and $x(\delta(w) \cap \delta(S)) + x(\delta(R_1) \cap \delta(S)) = x(\delta(S)) = f(S) = 1$. Therefore, $\delta(w, R_1) = \delta(R_1)$, and hence $\chi(\delta(w)) = \chi(\delta(R_1)) + \chi(\delta(S))$, contradicting the linear independence of these vectors.

(c) Suppose $deg(w) = 5$. Then by Step 2b of the algorithm, $B_w \leq 1$ and hence $B_w = 1$. Each child R_i has a heavy edge e_i incident on w. So S can have only one child R_1. We cannot have $\delta(w, R_1) = \delta(R_1)$; otherwise $B_w > 1$. Also, there is no heavy edge in $\delta(w) \cap \delta(S)$; otherwise $B_w > 1$. So w has at least five tokens. If S owns an endpoint, then S can collect one more token, as required. So further assume that S does not own an endpoint. Then $x(\delta(w) \cap \delta(S)) + x(\delta(w, R_1)) = x(\delta(w)) = B_w = 1$, and $x(\delta(w, R_1)) + x(\delta(R_1) \cap \delta(S))x(\delta(R_1)) = f(R_1) = 1$, and $x(\delta(w) \cap \delta(S)) + x(\delta(R_1) \cap \delta(S)) = x(\delta(S)) = f(S) = 1$. Therefore, $x(\delta(R_1) \cap \delta(S)) = x(\delta(w, R_1)) = x(\delta(w) \cap \delta(S)) = \frac{1}{2}$. Since $\delta(R_1) = 2$, there is only one edge $e \in \delta(R_1) \cap \delta(S)$, having $x_e = \frac{1}{2}$. So, e is an out-heavy edge of R_1 in $\delta(S)$, contradicting $h = 0$.

This completes the proof of Claim 5.16. □

We now show that the induction hypothesis holds when S does not own a vertex of W.

Base Case of Lemma 5.11: $S \in \mathcal{L}$ is a leaf node in the laminar family. S gets one token for each edge in $\delta(S)$ since S does not own a vertex in W. Therefore, it gets two tokens only if S is of Class I, three tokens only if it is of Class II, and at least four tokens in any other case, as required.

Induction step of Lemma 5.11: The proof is by induction on number of children of S. Let h be the number of out-heavy edges in S, and let t be the number of tokens that S can collect. In the following we say a child R is of *Type A* if R is of Class Ia or of Class IIa. Note that we need $h + t \geq 4$ if S is not of Type A, and $h + t \geq 3$ if S is of Type A. The following Claim 5.17 is crucial and needs the definition of out-heavy edges and Rule (2) of the token assignment scheme.

Claim 5.17 *Each Class Ib, Class IIb, or Class III child R of S can contribute at least 2 to $h+t$. Also each Class Ia, Class IIa child can contribute at least 1 to $h+t$.*

Proof: If R is of Class III, then it has 2 excess tokens. If R is of Class IIb, then it has 1 excess token and one out-heavy edge $e \in \delta(R)$. If $e \in \delta(S)$, then it contributes 1 to h. Otherwise if both endpoints of e are in S, then it contributes 1 to t by Rule (2) of the token assignment scheme. Note that, by definition, an edge can be an out-heavy edge of at most one child of S, and so its contribution to t will not be double counted. If R is of Class Ib, then it has 2 out-heavy edges. By the same argument, these edges contributes 2 to $h+t$. Similarly, if R is of Class Ia, then it has 1 out-heavy edge, and thus contributes 1 to $h+t$. Finally, if R is of Class IIa, then it has 1 excess token. □

For the remaining argument we use the following claim which follows from Jain [53] and is similar to Claim 5.8.

Claim 5.18 *[53] If S does not own any vertices in W and owns α tokens and has β children all of Class IIa and $\alpha+\beta=3$ then S is of Class IIa.*

We now prove a claim which helps us prove the various cases of the induction. The proofs are similar to proofs in Jain [53].

Claim 5.19 *Suppose S is a set which does not own any vertices in W, has $\alpha \geq 1$ children all of which are Type A, owns β endpoints and has no out-heavy edges in $\delta(S)$ for which one endpoint is owned by S. If $\alpha+\beta=3$ then S is of Type A.*

Proof: We prove the claim by a case analysis on different values of α.

1. $\alpha = 1$. Thus $\beta = 2$. Let R be the child of S. Since $\chi(\delta(R))$ and $\chi(\delta(S))$ are independent, there must exist edges $e \in \delta(R) \setminus \delta(S)$ and $f \in \delta(S) \setminus \delta(R)$ and S receives one token for both these edges. Moreover there is no other edge in $\delta(S) \setminus \delta(R)$ or $\delta(R) \setminus \delta(S)$ since $\beta = 2$. Now, if R is of Class Ia then the out-heavy edge in $\delta(R)$ must also be in $\delta(S)$ since S does not own a vertex in W. In this case S is also of Class Ia. If R is of Class IIa then $x_e = x_f < \frac{1}{2}$ and $\delta(S) = \delta(R) \cup \{f\} \setminus \{e\}$ and S is also of Class IIa.

2. $\alpha = 2$. Thus $\beta = 1$. Let R_1 and R_2 be the children of S. R_1 and R_2 cannot both be of Class Ia since the out-heavy edge in $\delta(R_i)$ must be in $\delta(S)$ for $i = 1, 2$, but

then $x(\delta(S)) > 1$, a contradiction. First suppose R_1 is of Class Ia and therefore R_2 is of Class IIa. We cannot have $|\delta(R_1, R_2)| \geq 2$ since $|\delta(R_1)| = 2$, and also cannot have $|\delta(R_2) \cap \delta(S)| \geq 2$ since $f(S) = x(\delta(S)) = 1$. So the only possibility is $|\delta(R_1, R_2)| = |\delta(v, R_2)| = |\delta(R_2) \cap \delta(S)| = 1$. Hence $|\delta(S)| = 2$ and thus S is of Class Ia, as required. Finally, suppose both R_1 and R_2 are of Class IIa and $\delta(S)$ does not contain any heavy edges, then from Claim 5.18 S is of Class IIa, as required.

3. $\alpha = 3$. Let R_1, R_2 and R_3 be the children of S. As previously argued in the case of $\alpha = 2$, at most one of R_1, R_2, R_3 can be of Class Ia. First suppose that S has exactly one Class Ia child, say R_1. Let $\delta(R_1) = \{e_1, f_1\}$, where e_1 is the out-heavy edge of R_1. Assume, without loss of generality, that $f_1 \in \delta(R_2)$. Since $f(S) = 1$, we must have $|\delta(R_2, R_3)| = 2$; otherwise $|\delta(R_3) \cap \delta(S)| \geq 2$ and thus $x(\delta(S)) = x_{e_1} + x(\delta(R_3) \cap \delta(S)) > \frac{1}{2} + \frac{1}{2} = 1$, since $|\delta(R_3)| = 3$ and each edge e in $\delta(R_3)$ has $x_e < \frac{1}{2}$ by the definition of a Class IIa child. Since $|\delta(R_2, R_3)| = 2$, this implies that $\delta(S) = 2$, and hence S is of Class Ia and therefore Type A. In the other case, we have that all three children of S are of Class IIa. Then from Claim 5.18 it follows that S must also be of Class IIa.

Thus the claim follows. □

Now we complete the inductive argument based on the number of children of S.

1. S has at least four children. Then each child can contribute at least 1 to $h + t$, and so $h + t \geq 4$.

2. S has exactly three children. If there is a child which is not of Type A, then $h+t \geq 4$ by Claim 5.17, as required. So assume S has exactly three Type A children R_1, R_2, R_3. If S owns an endpoint then also $h + t \geq 4$. So further assume that S does not own an endpoint. Then S satisfies the conditions of Claim 5.19 and must be of Type A. Thus $h + t \geq 3$ suffices for S.

3. S has exactly two children R_1 and R_2. If both R_1 and R_2 are not of Type A, since each can contribute 2 to $h + t$ by Claim 5.17, then we are done.

   Suppose R_1 is of Type A and R_2 is not of Type A. If S owns an endpoint then we are done. So further assume that S does not own an endpoint. We shall prove that

this would not happen. In this case

$$\begin{aligned} x(\delta(R_1) \cap \delta(S)) + x(\delta(R_1, R_2)) &= x(\delta(R_1)) = 1, \\ x(\delta(R_1) \cap \delta(S)) + x(\delta(R_2) \cap \delta(S)) &= x(\delta(S)) = 1, \\ x(\delta(R_2) \cap \delta(S)) + x(\delta(R_1, R_2)) &= x(\delta(R_2)) = 1, \end{aligned}$$

Thus we have,

$$x(\delta(R_1) \cap \delta(S)) = x(\delta(R_1, R_2)) = x(\delta(R_2) \cap \delta(S)) = \frac{1}{2}.$$

R_1 cannot be of Class Ia, since otherwise it has an edge with $x_e > \frac{1}{2}$. Also, R_1 cannot be of Class IIa, since $|\delta(R_1)| = 3$, either $\delta(R_1, R_2)$ or $\delta(R_2) \cap \delta(S)$ is a single edge e with $x_e = \frac{1}{2}$, contradicting R_2 is of Class IIa.

So suppose R_1 and R_2 are of Type A. If S owns two endpoints, then we are done. By the above argument, S must owns at least one endpoint, and thus $h + t \geq 3$. If S has an out-heavy edge for which S owns one endpoint then we have $h + t \geq 4$ and we are done. Hence assume that S owns exactly one endpoint v and each out-heavy edge in $\delta(S)$ is in $\delta(R_i)$ for some i. Thus S satisfies the condition of Claim 5.19 and is of Type A. Thus $t \geq 3$ suffices for S.

4. S has exactly one child R. By linear independence of $\chi(\delta(S))$ and $\chi(\delta(R))$, S must own at least two endpoints, and thus $h+t \geq 3$. If R is not of Type A, then $h+t \geq 4$, and we are done. If $\delta(S) \setminus \delta(R)$ has an out-heavy edge or S owns more than two endpoints then also we have $h + t \geq 4$ as required. In the remaining case S satisfies conditions of Claim 5.19 and S is of Type A. Therefore, $h + t \geq 3$ suffices.

This completes the proof of Lemma 5.11. If some root S of the laminar family is not of Class I, then there is some excess token left at S by Lemma 5.11. If every root is of Class I, then there must exist a vertex $w \in W$ that is not contained in any root, and so there is some excess token left at w. This completes the proof of Theorem 5.9. □

6

Directed Network Design with Degree Constraints

In this chapter we present bi-criteria approximation algorithms for bounded-degree network design problems in directed graphs.

- In section 6.1, we consider bounded-degree network design problem in directed graphs where the connectivity requirements can be specified by a *crossing supermodular function*. An example of such network design problems is the k-arc connected spanning subgraph problem. We give a $(2, 3B + 5)$-approximation algorithm for the problem.

- In section 6.2, we consider the special case of bounded-degree network design problem in directed graphs here the connectivity requirements are specified by a $\{0, 1\}$-valued *intersecting supermodular* function. This family of problems includes the Minimum Bounded-Degree Arborescence problem. We give an improved $(2, 2B + 2)$-approximation algorithm for this case.

6.1 Minimum Degree-Bounded Directed Networks

In this section, we consider a general network design problem with degree constraints where the connectivity requirement is given by a crossing supermodular function. An integer function on sets of vertices $f : 2^V \rightarrow \mathbb{Z}^+$ is called *crossing supermodular* if the inequality

$$f(A) + f(B) \leq f(A \cap B) + f(A \cup B)$$

holds for every pair of sets $A, B \subseteq V$ such that $A \cap B \neq \emptyset$ and $A \cup B \neq V$. The connectivity requirement of *k-edge-connected spanning subgraph* can be formulated via the crossing supermodular function $f(S) = k$, $\forall \emptyset \neq S \subsetneq V$.

$$
\begin{array}{lrcll}
\text{(DLP) minimize} & z_{DLP} & = & \displaystyle\sum_{e \in E} c_e\, x_e & \\
\text{subject to} & \displaystyle\sum_{e \in \delta^{in}(S)} x_e & \geq & f(S), & \forall\, S \subseteq V, r \notin S \\
& \displaystyle\sum_{e \in \delta^{in}(v)} x_e & \leq & B_v^{in}, & \forall\, v \in W_1 \\
& \displaystyle\sum_{e \in \delta^{out}(v)} x_e & \leq & B_v^{out}, & \forall\, v \in W_2 \\
& 0 \leq x_e & \leq & U_e, & \forall\, e \in E
\end{array}
$$

Figure 6.1: Linear Program for Directed Connectivity.

First we address the MINIMUM BOUNDED-DEGREE DIRECTED NETWORK DESIGN problem of finding a minimum cost subgraph satisfying crossing supermodular connectivity requirements f and degree bounds B_v^{in} and B_v^{out} on the in-degree and out-degree, respectively, for each $v \in V$. We prove the following theorem.

Theorem 6.1 *There is a polynomial time algorithm which given an instance of the* MINIMUM BOUNDED-DEGREE DIRECTED NETWORK DESIGN *problem returns a solution H of cost $\leq 3c(OPT)$ and $\delta_H^{out}(v) \leq 3B_v^{out} + 5$ and $\delta_H^{in}(v) \leq 3B_v^{in} + 5$ for all $v \in V$ where OPT is the optimal solution.*

6.1.1 Linear Program

Figure 6.1 shows the linear program (DLP) for the problem. As before U_e is the upper bound on the multiplicity of edge e. We place out-degree bounds for vertices in $W_1 \subseteq V$ and in-degree bounds for vertices in $W_2 \subseteq V$ both of which can be initialized to V initially.

The linear program (DLP) can be optimized in polynomial time (see Frank [35] and Melkonian-Tardos [101]).

6.1.2 Characterization of Vertex Solutions

Recall that a pair of sets A, B are *crossing* if all of the sets $A \cap B, A - B, B - A, V - (A \cup B)$ are nonempty, and a family of sets $\mathcal{L} = \{A_1, A_2, \ldots, A_\ell\}$ is *cross-free* if no two of its sets are crossing. For any set $A \subseteq V$, let $\chi(A)$ denote the incidence vector of the set of arcs

$\delta^{in}(A)$.

By an extension of a result in Frank [35] and Melkonian-Tardos [101] the following lemma is immediate.

Lemma 6.2 *Let the requirement function f be crossing supermodular, and let x be a vertex solution of the linear program (DLP) such that $0 < x_e < 1$ for all edges $e \in E$. Then there exists a* ***cross-free family*** *$\mathcal{Q}$ of tight sets and tight degree constraints for $T_1 \subseteq W_1$ and $T_2 \subseteq W_2$ such that*

(i) $|\mathcal{Q}| + |T_1| + |T_2| = |E|$.

(ii) The vectors $\chi(A)$ for $A \in \mathcal{Q}$, $\chi(v)$ for $v \in T_1$ and $\chi(V \setminus v)$ for $v \in T_2$ are linearly independent.

(iii) x is the unique solution to $\{x(\delta^{in}(v)) = B_v^{in}, \forall v \in T_1\} \bigcup \{x(\delta^{out}(v)) = B_v^{out}, \forall v \in T_2\} \bigcup \{x(\delta^{in}(A)) = f(A), \forall A \in \mathcal{Q}\}$.

6.1.3 Iterative Algorithm

The algorithm is presented in Figure 6.2. The following lemma ensures that we always make progress either in Step 2b or Step 2c. Observe that the proof of Theorem 6.1 follows from Lemma 6.3.

Lemma 6.3 *Given a vertex solution x of (DLP) in Figure 6.1 where f is a crossing supermodular function, one of the following conditions must be true.*

1. *There exists $v \in W_1$ with $|\delta^{in}(v)| \leq 6$,*
2. *There exists $v \in W_2$ with $|\delta^{out}(v)| \leq 6$,*
3. *There exists an edge e such that $x_e \geq \frac{1}{3}$.*

To prove Lemma 6.3, we first introduce some notation and preliminaries. The cross free family $\mathcal{Q}$ corresponds to a laminar family $\mathcal{L} = \mathcal{I} \cup \mathcal{O}$ with $|\mathcal{L}| = |\mathcal{Q}|$ such that $x(\delta^{in}(S)) = f(S)$ for each $S \in \mathcal{I}$ and $x(\delta^{out}(S)) = x(\delta^{in}(V - S)) = f(V - S)$ for each $S \in \mathcal{O}$ (see Melkonian-Tardos [101]). Also, we augment the family $\mathcal{L}$ by including in it singleton sets corresponding to tight degree constraints in T_1 and T_2 to obtain $\mathcal{L}' = \mathcal{I}' \cup \mathcal{O}'$

1. Initialization $F \leftarrow \emptyset$, $f' \leftarrow f$, and $\forall v \in W_1$: $B'^{in}_v = B^{in}_v$ and $\forall v \in W_2 : B'^{out}_v = B^{out}_v$.
2. While $f' \neq \emptyset$ do
   (a) Find a vertex solution x with cut requirement f' and remove every edge e with $x_e = 0$.
   (b) If there exists a vertex $v \in W_1$ with indegree at most 6, remove v from W_1; if there exists a vertex $v \in W_2$ with outdegree at most 6, remove v from W_2. Goto (a).
   (c) For each edge $e = (u, v)$ with $x_e \geq 1/3$, add e to F and decrease B'^{out}_u and B'^{in}_v by 1/3.
   (d) For every $S \subseteq V$: $f'(S) \leftarrow f(S) - |\delta^{in}_F(S)|$.
3. Return $H = (V, F)$.

Figure 6.2: Bounded-Degree Directed Graph Algorithm.

where $\mathcal{I}' = \mathcal{I} \cup \{v\}_{v \in T_1}$ and $\mathcal{O}' = \mathcal{O} \cup \{v\}_{v \in T_2}$. Observe that $|\mathcal{L}'| = |\mathcal{Q}| + |T_1| + |T_2| = |E|$. We call members of $\mathcal{I}'$ *square* sets and members of $\mathcal{O}'$ *round* sets.

We now prove Lemma 6.3. The proof is an extension of a similar result (Theorem 3.1) in Gabow [38] where the existence of an edge $x_e \geq \frac{1}{3}$ is proved when degree constraints are not present. In the presence of degree constraints we show that either we have an edge with $x_e \geq \frac{1}{3}$ or the condition where a degree constraint is removed in Lemma 6.3 is satisfied. The laminar family $\mathcal{L}'$ corresponds to a forest $\mathcal{F}$ over the sets in the laminar family where $B \in \mathcal{L}'$ is a child of $A \in \mathcal{L}'$ if A is the smallest set containing B. A node A of $\mathcal{L}'$ is a *leaf, chain node* or *branching node* depending on whether it has $0, 1$ or > 1 children. A chain node is a 1-chain node (or 1-node) if it belongs to same family $\mathcal{I}'$ or $\mathcal{O}'$ as its unique child, else it is a 2-chain node (2-node).

Proof of Lemma 6.3: The proof is by contradiction. Suppose neither of the three conditions holds. We show this leads to the contradiction to the fact that $|\mathcal{Q}|+|T_1|+|T_2| = |\mathcal{L}'| = |E|$. The argument proceeds by assigning two tokens for each edge (one to each endpoint of e), and showing by a counting argument that we can collect two tokens for each member in the laminar family and are still left with some excess tokens.

The token assignment is a detailed argument following Gabow [38] depending on the different cases of the sets. We point out some simple cases from the argument in Gabow [38]

and where the presence of degree constraints leads us to give a different argument.

Firstly, we give the following definitions following Gabow [38]. Consider a chain node S with unique child A. Let e be an edge with an end in $S \setminus A$. We will call e *p-directed* (for parent-directed) if it is oriented consistent with S's family ($\mathcal{I}'$ or $\mathcal{O}'$). We call e *c-directed* (for child-directed) if it is oriented consistent with A's family.

The following rule is used to assign the token for endpoint v of edge e.

Definition 6.4 *The token for the endpoint v of an edge e is given to node S of $\mathcal{L}'$ if one of the following holds:*

1. *When S is a leaf, $v \in S$, and e is directed consistent with S's family, i.e., either $S \in \mathcal{I}'$ and $e \in \delta^{in}(S)$ or $S \in \mathcal{O}'$ and $e \in \delta^{out}(S)$.*
2. *When S is a 1-chain node, $v \in S \setminus A$ for A child of S and e is p-directed (or equivalently, c-directed).*

Observe that each leaf node which corresponds to a degree constraint obtains at least 7 tokens otherwise the degree constraint can be removed. They only need two tokens for themselves for the counting argument. The five extra tokens are assigned to other nodes in three different steps, the first of which is the the following lemma.

Lemma 6.5 *The number of ends available to leaves of $\mathcal{L}'$ can be redistributed to give two tokens to each leaf and branching node of $\mathcal{L}'$ and five tokens to each leaf node which is a degree constraint.*

Proof: A leaf node not corresponding to a degree constraint gets at least four tokens, for example, $S \in \mathcal{I}$ receives one token for each edge $e \in \delta^{in}(S)$ and $|\delta^{in}(S)| \geq 4$ since $x(\delta^{in}(S)) = f(S) \geq 1$ and there is no edge e with $x_e \geq \frac{1}{3}$. Leaf nodes which correspond to degree constraint receive at least seven tokens. Since, the number of branching nodes in any tree is strictly less than the number of leaves, we can assign two tokens from each of the leaves to branching nodes giving the claim. □

Now, we still have three extra tokens with the sets corresponding to the degree constrained leaves, one of which we use in the following lemma.

Lemma 6.6 *Each 1-chain node has at least two available ends if each set in $\mathcal{L}'$ corresponding to a degree constraint donates one token.*

Proof: Consider a 1-chain node S with a child A where wlog $S, A \in \mathcal{I}'$. If both $S, A \in \mathcal{I}$ then we have $x(\delta^{in}(S)) = f(S)$ and $x(\delta^{in}(A)) = f(A)$. Observe that each edge in the difference with a non-zero ($+1$ or -1) co-efficient gives one token to S. Independence of the constraints implies that there must be at least one such edge and the integrality of $f(S)$ and $f(A)$ implies that there cannot be exactly one such edge. Hence, S obtains two tokens in this case.

In the other case, we may have that A corresponds to a degree constraint. Then we do not have integrality since $x(\delta^{in}(A)) = B_v^{in}$ where $A = \{v\}$ and B_v^{in} need not be an integer. But, independence of constraints implies that S receives at least one token and it borrows another token from A for the induction claim to hold. □

The rest of the proof involves analysis of 2-nodes. Lemma 6.3 follows from Lemma 6.5 and Lemma 6.6 if we can show that 2-chains can collect two tokens each for themselves from the remaining unassigned tokens and two extra tokens with each degree constraint.

We start the analysis by defining a subtree $\mathcal{F}_S$ for each 2-chain node S. $\mathcal{F}_S$ is the minimal subtree of $\mathcal{F}$ having S as its root and each leaf either a leaf of $\mathcal{L}'$ or a 2-chain node other than S. In particular, S is always a internal node of tree $\mathcal{F}_S$ and not a leaf node.

The various trees $\mathcal{F}_S$ can overlap: A 2-chain node S occurs at the root in $\mathcal{F}_S$ and also as a leaf in $\mathcal{F}_T$ for T the first 2-chain node that is a proper ancestor of S. It is easy to see that these are the only 2-possibilities. Also observe that a set corresponding to a degree constraint can only occur in one tree since it can never be a root in such a tree.

The token assignment is as follows. Each set A corresponding to a degree constraint gives two its excess tokens to the 2-chain node S where $A \in \mathcal{F}_S$. Thus, each 2-chain node S receives two tokens whenever there is a degree constraint in $\mathcal{F}_S$. In the remaining case we have that there is no degree constraint in $\mathcal{F}_S$. Then we use the following token assignment from Gabow [38] (pages 120-125) which states that 2-chain nodes can pay for themselves.

Lemma 6.7 *[38] The total number of tokens available for the* 2*-chain nodes in the above assignment equals twice the number of such nodes.*

We omit the proof of the lemma. This completes the proof of Lemma 6.3. ■

6.2 Minimum Bounded-Degree Arborescence

We now show how to improve the bounds in the case of intersecting supermodular connectivity requirements. An integer function on sets of vertices $f : 2^V \rightarrow \mathbb{Z}^+$ is called intersecting supermodular if the inequality

$$f(A) + f(B) \leq f(A \cap B) + f(A \cup B)$$

holds for every pair of sets $A, B \subseteq V$ such that $A \cap B \neq \emptyset$. This is a stronger requirement than crossing supermodularity; for example the connectivity requirements of a strongly k-edge-connected subgraph cannot be formulated as an intersecting connectivity requirement function since when $A \cup B = V$, $f(A \cup B) = 0$. The intersecting supermodular connectivity requirement nonetheless captures the problem of finding an arborescence rooted at r where $f(S) = 1$ if $r \notin S$ and 0 otherwise. The linear programming relaxation is identical to the linear program in Figure 6.1. We prove the following theorem.

Theorem 6.8 *There exists a polynomial time algorithm which, given a directed graph G, a $\{0,1\}$-valued intersecting supermodular function f as the connectivity requirement and degree bounds B_v^{in} and B_v^{out} for each vertex, returns a solution H of cost $\leq 2 \cdot c(OPT)$ where $c(OPT)$ is the cost of the optimum solution. Moreover, $deg_H^{out}(v) \leq 2B_v^{out} + 2$ and $deg_H^{in}(v) \leq 2B_v^{in} + 2$ for all $v \in V$.*

Before we prove Theorem 6.8 we give a stronger characterization of vertex solutions than in Section 6.1.2.

6.2.1 Linear Program

We use the same linear program (DLP) in Figure 6.1 with the special case that the function f is intersecting supermodular. Since, intersecting supermodular functions are special cases of crossing supermodular function, polynomial time solvability follows from the discussion in the previous section.

6.2.2 Characterization of Vertex Solutions

The following lemma is immediate from Frank [35] and Lemma 6.2.

Lemma 6.9 *Let the requirement function f be intersecting supermodular, and let x be a vertex solution of (DLP) such that $0 < x_e < 1$ for all edges $e \in E$. Then there exists a* ***laminar family*** *$\mathcal{Q}$ of tight sets and tight degree constraints for $T_1 \subseteq W_1$ and $T_2 \subseteq W_2$ such that*

(i) $|\mathcal{Q}| + |T_1| + |T_2| = |E|$.

(ii) The vectors $\chi(A)$ for $A \in \mathcal{Q}$, $\chi(v)$ for $v \in T_1$, and $\chi(V \setminus v)$ for $v \in T_2$ are all linearly independent.

(iii) x is the unique solution to $\{x(\delta^{in}(v)) = B_v^{in}, \forall v \in T_1\} \bigcup \{x(\delta^{out}(v)) = B_v^{out}, \forall v \in T_2\} \bigcup \{x(\delta^{in}(A)) = f(A), \forall A \in \mathcal{Q}\}$.

Observe that Lemma 6.9 differs from Lemma 6.2 in the fact that, in case of a intersecting supermodular function, we can ensure that the independent set of inequalities correspond to a laminar family while in the case of a crossing supermodular function we could only ensure that the independent set of inequalities correspond to a cross-free family.

6.2.3 Iterative Algorithm

The algorithm is identical to the algorithm in Figure 6.2 except that in Step 2c, we only pick an edge e if $x_e \geq \frac{1}{2}$ and decrease the degree bounds by 1/2, and also in Step 2b, we remove an indegree or outdegree constraint if a vertex's indegree or outdegree is at most 3. We remark that the half-integrality of degree bounds will be useful later. We now prove the following lemma which gives Theorem 6.8.

Lemma 6.10 *Given a vertex solution x of (DLP) in Figure 6.1 where f is an intersecting supermodular function, one of the following must be true.*

1. *There exists $\{v\} \in T_1$ with $|\delta^{in}(v)| \leq 3$,*
2. *There exists $\{v\} \in T_2$ with $|\delta^{out}(v)| \leq 3$,*
3. *There exists an edge e such that $x_e \geq \frac{1}{2}$.*

Proof: Suppose neither of the above conditions hold. Then each vertex with a tight in-degree constraint must have at least four in-edges and a vertex with a tight out-degree constraint must have at least four out-edges and each edge e must have $x_e < \frac{1}{2}$. Now,

we argue this leads to a contradiction to the fact that $|\mathcal{Q}| + |T_1| + |T_2| = |E|$. We prove this by the following counting argument. For each edge we assign three tokens. We then redistribute these tokens such that each constraint gets assigned at least three tokens and we still have extra tokens.

In the initial assignment, each edge gives one token to the head and two tokens to the tail of the edge. Hence each vertex gets two tokens for each out-edge incident at it and one token for each in-edge incident at it. For a vertex $v \in T_2$, we use one token for each out-edge at v for the out-degree constraint of v. We use rest of the tokens for in-degree constraints and connectivity constraints.

Observe that each vertex with an out-degree constraints must have at least four out-edges incident at it. Hence, when we take one token for each out-edge, we obtain at least four tokens for the out-degree constraint, i.e., they have one excess tokens.

For each vertex, we have one token for each in-edge and out-edge incident at it remaining. Moreover, if $v \notin T_2$ we still have two tokens for each out-edge incident at v. We re-assign these tokens such that we collect at least three tokens for each tight in-degree constraint in T_1 and three tokens for each connectivity constraint in $\mathcal{Q}$. We outline this below.

Observe that $\mathcal{Q}' = \mathcal{Q} \cup \{v\}_{v \in T_1}$ is a laminar family where we have the constraints $x(\delta^{in}(S)) = g(S)$ where $g(S) = f(S)$ if $S \in \mathcal{Q}$ and $g(S) = B_v^{in}$ if $S = \{v\}$ for $v \in T_1$. For the laminar family $\mathcal{Q}'$, let $\mathcal{L}'$ be the forest on the members of the laminar family. Observe that each member of T_1 in $\mathcal{L}'$ is a leaf and any non-leaf node corresponds to a connectivity constraint. We use this fact crucially when we use the fact the connectivity requirements are integral. We say that a vertex v is owned by $S \in \mathcal{Q}$ if S is the smallest set in $\mathcal{Q}$ containing v. Now, we prove the following lemma.

Lemma 6.11 *Given a subtree of $\mathcal{L}'$ rooted at S, we can assign three tokens to each tight-degree constraint in S and three tokens to each set R in the subtree. Moreover, we can assign $3 + |\delta^{out}(S)|$ tokens to the root S if S corresponds to a connectivity constraint and $4 + |\delta^{out}(S)|$ tokens if S corresponds a degree constraint.*

Proof: The proof is by induction on the height of the subtree.

Base Case. S is a leaf. If S corresponds to a tight-indegree constraint, we must have four in-edges incident at S. Else if S corresponds to a tight connectivity constraint we have $x(\delta^{in}(S)) = f(S)$ where $f(S)$ is a positive integer. The assumption that there is no edge with $x_e \geq \frac{1}{2}$ implies that there must be three edges in $\delta^{in}(S)$. For each out-edge

incident at S, S can collect one token. Hence, there must be at least three in-edge tokens (four if S corresponds to a tight degree constraint) and $|\delta^{out}(S)|$ out-edge tokens which can be assigned to S.

Induction Case. S is not a leaf. Hence, S must correspond to a tight connectivity constraint. By induction, we assign $3+|\delta^{out}(R)|$ tokens to each child R of S ($4+|\delta^{out}(R)|$ if R corresponds to a tight degree constraint). Each child R of S donates one token to S for each edge in $\delta^{out}(R)$. First observe that we can assign one token to S for each out-edge $e \in \delta^{out}(S)$. If the tail of e is in some child R of S, then R has already donated one token for this edge. Else, the tail has been assigned one token for this edge in the initial assignment and can give one token to S. Thus S can be assigned one token for each edge in $\delta^{out}(S)$.

Case 1. S has at least two children $R_1, R_2 \in \mathcal{Q}$. Since each tight set has connectivity requirement exactly 1, we have $\sum_{R\in\mathcal{Q}} f(R) - f(S) \geq 1$. Let $F_1 = \delta^{in}(S) \setminus (\cup_R \delta^{in}(R))$ and $F_2 = (\cup_R \delta^{in}(R)) \setminus \delta^{in}(S)$. The above inequality implies that $x(F_2) \geq 1$. But then we have $|F_2| \geq 3$, as there is no edge e with $x_e \geq \frac{1}{2}$. So S can collect one token for each edge in F_1 (token assigned to head) and F_2 (one of the two tokens assigned to tail) to get three tokens.

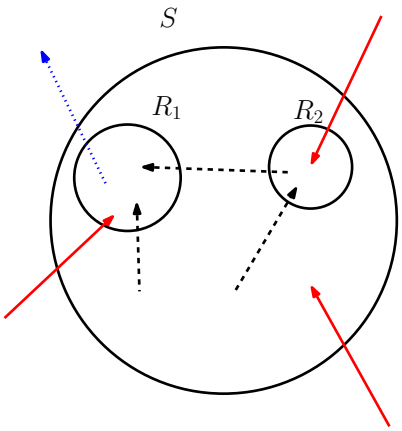


Figure 6.3: Here the solid edges are in F_1, dashed edges are in F_2 and the dotted edges are in $\delta^{out}(S)$. S can collect one token for each edge in F_1 and F_2.

Case 2. S has exactly one child $R \in \mathcal{Q}$. Since $f(S) = f(R)$ and $\chi(S)$ and $\chi(R)$ are linearly independent, we have $|F_1| \geq 1$ and $|F_2| \geq 1$, where F_1 and F_2 are defined as in previous case. So S can collect one token for each edge in F_1, and one token for each edge in F_2. If S also has a child which is a degree constraint, then we can also collect one excess token from it. Otherwise, the tail of any edge in F_2 does not have a tight degree constraint,

and thus can contribute two tokens to S. In either case S can collect the desired three tokens.

Case 3. S has no child in $\mathcal{Q}$. Hence, each child of S must be in T_1. If S has at least two children or if S owns a vertex in T_2 then S can collect at least one excess token from each child in T_1 and each vertex in T_2 owned by S. Moreover, it can collect one more token from either an edge in F_1 or in F_2 by linear independence (where F_1 and F_2 are defined as before), and we are done.

Thus the only case left is that S has only one child R which is a singleton vertex from T_1 and S does not own any vertices in T_2. S can collect one excess token from R and needs two more. If $f(S) = g(R)$ (Here $g(R) = B_v$ where $R = \{v\}$), by the argument in case 2, $|F_1| \geq 1$ and $|F_2| \geq 1$, and S can collect two more tokens. If $f(S) \neq g(R)$, then $|f(S) - g(R)|$ is half-integral since $f(S)$ is an integer and $g(R)$ is half-integral. Therefore, since there is no edge e with $x_e \geq \frac{1}{2}$, either $|F_1| \geq 2$ or $|F_2| \geq 2$, and thus S can collect two more tokens. □

Lemma 6.11 reassigns the tokens such that we have three tokens for each member in $\mathcal{Q}$ and three tokens for each vertex in T_1 and T_2. To prove Lemma 6.10 it remains to be shown that some tokens are still left in excess. If any root S of the forest $\mathcal{L}'$ has at least one out-edge, then S has been assigned at least four tokens and one excess token with S gives us the contradiction. Else, consider any root S. Any $e \in \delta^{in}(S)$ must have its tail at a vertex not owned by any set in $\mathcal{Q}'$. If the tail of e has an out-degree constraint present, it has at least one excess token. Else the out-token for e has not been used in the above assignment and is the excess token which gives us the contradiction. This proves Lemma 6.10. □

7

Further Applications

In this chapter, we consider more applications of the iterative method. We will extend the integrality results developed in Chapter 3 and show approximation algorithms for three problems. The first problem is the generalized assignment problem where we present an alternate proof of the result of Shmoys and Tardos [95] achieving a 2-approximation. We then give a PTAS for multi-criteria spanning trees and multi-criteria matroids. We then consider generalizations of the minimum bounded degree spanning tree problem to degree constrained matroids.

7.1 Generalized Assignment

In the section, we use the iterative relaxation method to obtain an approximation algorithm for the generalized assignment problem. The generalized assignment problem models the problem of scheduling jobs on unrelated parallel machines with costs is defined as follows. We are given a set of Jobs J and machines M, for each job j and machine i there is a processing time p_{ij} and cost c_{ij}; each machine i is available for T_i time units and the objective is to assign each job to some machine such that the total cost is minimized and no machine is scheduled for more than its available time. A job j assigned to machine i is said to *use* time p_{ij} on machine i. A machines is said to used for time T if all the jobs assigned to it use a total time of T.

Shmoys and Tardos [95] gave an algorithm which returns an assignment of cost at most C and each machine is used at most $2T_i$ time units where C is the cost of the optimal assignment which uses machine i for at most T_i time units (if such an assignment is possible). In this section, we prove the result of the Shmoys and Tardos [95] using the iterative relaxation method. This proof develops on the iterative proof of the integrality

of the linear program $LP_{BM}(G)$ for the bipartite matching matching given in Section 3.1. We prove the following theorem.

Theorem 7.1 *[95] There exists a polynomial time algorithm which, given an instance of the generalized assignment problem, returns a solution of cost at most C and any machine i is used for $2T_i$ time units where C is the cost of the optimal assignment which uses machine i for at most T_i units.*

7.1.1 Linear Programming Relaxation

Before we write the linear program for the problem, we first model the problem as a matching problem. We start with a complete bipartite G with jobs J and machines M as the two sides of the bipartite graph. The edge between job j and machine i has cost c_{ij}. The generalized assignment problem can be reduced to finding a subgraph F of G such that $deg_F(j) = 1$ for each job $j \in J$. The edge incident at j denotes to which machine job j is assigned. The time constraint at machines can be modelled by specifying that $\sum_{e \in \delta(i) \cap F} p_{ij} \leq T_i$ for each machine i. We now strengthen this model by disallowing certain assignments using the following observation. If $p_{ij} > T_i$ then no optimal solution assigns job j to i. Hence, we remove all such edges from graph G.

We now model the matching problem by the following natural linear programming relaxation $LP_{GA}(G, M')$ where $M' \subseteq M$ to prove Theorem 7.1. Observe that we do not place time constraints for all machines but a subset $M' \subseteq M$ which can be initialized to M. We have a variable x_e for each $e = \{i, j\}$ denoting whether job j is assigned to machine i.

$$\text{minimize} \quad c(x) = \sum_{e \in E} c_e x_e \tag{7.1}$$

$$\text{subject to} \quad \sum_{e \in \delta(j)} x_e = 1 \qquad \forall j \in J \tag{7.2}$$

$$\sum_{e \in \delta(i)} p_e x_e \leq T_i \qquad \forall i \in M' \tag{7.3}$$

$$x_e \geq 0 \qquad \forall e \in E \tag{7.4}$$

7.1.2 Characterization of Vertex solutions

The following lemma follows from a direct application of the Rank Lemma.

Lemma 7.2 *Let x be an optimum vertex solution to the linear program $LP(G, M')$ such that $0 < x_{ij} < 1$ for each $\{i, j\} \in E(G)$. Then there exists a subset $J' \subseteq J$ and $M'' \subseteq M'$ such that*

1. *$\sum_{e \in \delta(j)} x_e = 1$ for each $j \in J'$ and $\sum_{e \in \delta(i)} p_e x_e = T_i$ for each $i \in M''$ and x is the unique solution to the these equalities.*
2. *The constraints corresponding to J' and M'' are linearly independent.*
3. *$|J'| + |M''| = |E(G)|$*

7.1.3 Iterative Algorithm

The following is a simple iterative procedure which returns a assignment of optimal cost. Observe that it generalizes the iterative procedure for finding optimal bipartite matchings in Section 3.1. The bipartite graph F with vertex set in $M \cup J$ returns the assignment found by the algorithm.

Iterative Generalized Assignment Algorithm

1. Initialization $E(F) \leftarrow \emptyset$, $M' \leftarrow M$.
2. While $J \neq \emptyset$
   (a) Find a vertex optimal solution x of $LP_{GA}(J, M, M')$ and remove every variable x_{ij} with $x_{ij} = 0$.
   (b) If there is a variable $x_{ij} = 1$ such that then update $F \leftarrow F \cup \{ij\}$, $J \leftarrow J \setminus \{j\}$, $T_i \leftarrow T_i - p_{ij}$.
   (c) Let $J_i = \{j \in J : x_{ij} > 0\}$ for each $i \in M$. If there is a machine i such that $|J_i| = 1$ or a machine i such that $|J_i| = 2$ and $\sum_{j \in J_i} x_{ij} \geq 1$ then update $M' \leftarrow M' \setminus \{i\}$.
3. Return F.

Figure 7.1: Generalized Assignment Algorithm.

The following lemma is the crucial lemma which shows that the algorithm makes progress at each step of the algorithm.

Lemma 7.3 *Consider any vertex solution x of $LP_{GA}(G, M')$ with support E. One of the following must hold.*

1. *There exists $e \in E$ such that $x_e \in \{0, 1\}$.*
2. *There exists an $i \in M'$ such that $deg_E(i) = 1$ or $deg_E(i) = 2$ and $\sum_{e \in \delta(i)} x_e \geq 1$.*

Proof: Suppose for the sake of contradiction both the conditions do not hold. Then each $0 < x_e < 1$. Each job $j \in J$ in the graph G has degree at least two as $\sum_{e \in \delta(j)} x_e = 1$. Moreover, each machine in M' has degree at least two, else there is a machine i such that $deg_E(i) = 1$. Hence, the total number of edges is at least $|M'| + |J|$. From Lemma 7.2 we have that $|E| = |J'| + |M''|$. But this possible only if $J = J'$ and $M' = M''$. Moreover, if there is a machine $i \in M \setminus M'$ with some edge incident at it then also we have a contradiction since then $|E| > |M'| + |J|$.

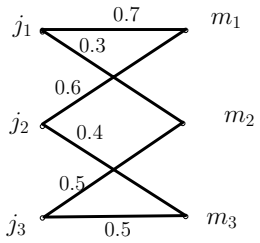


Figure 7.2: Here $\{j_1, j_2, j_3\}$ are the jobs assigned fractionally to $\{m_1, m_2, m_3\}$. Observe that m_1 is a machine to which two jobs are assigned fractionally and $x_{j_1 m_1} + x_{j_2 m_1} \geq 1$ as required.

Hence, each job and each machine in M' must have degree exactly two and there must be no edges incident at any machine in $M \setminus M'$. Hence, G is a union of cycles. Consider any cycle C in the graph G. The total number of jobs in C is exactly equal to the total number of machines in C. All jobs in C are completely assigned (though fractionally) to machines in C. Hence, there must be a machine i which is assigned exactly two jobs fractionally, i.e., $deg_E(i) = 2$ and $\sum_{j \in \delta(i)} x_{ij} \geq 1$. This is a contradiction to part 2 of condition 2. □

We now show the proof of Theorem 7.1 by a simple induction argument.

Proof of Theorem 7.1: We first prove that the algorithm returns an assignment of optimal cost. We claim that at any iteration of the algorithm the cost of assignment given by F plus the cost of the current linear programming solution to $LP_{GA}(G, M')$ is at most the cost of the initial linear programming solution. This we show by a simple inductive argument on the number of iterations. Observe that the claim holds trivially

before the first iteration. In any iteration, if we assign job j to machine i in Step 2b then the cost of F increases by c_{ij} and the current linear programming solution decreases by $c_{ij}x_{ij} = c_{ij}$ as $x_{ij} = 1$. Hence, the claim holds true. If we remove a constraint in Step 2c the cost of F remains same while we relax the linear program further and the cost of the current linear program can only decrease. Hence, in this case as well the claim holds. Thus, finally when F is a feasible assignment, the cost of assignment given by F is at most the cost of the initial linear programming solution which is at most C.

Now, we show for each i that machine i is used at most $2T_i$ units. Fix any machine i. We first argue the following claim. If $i \in M'$, then at any iteration we must have $T'_i + T_i(F) \leq T_i$ where T'_i is the residual time left on the machine at this iteration and $T_i(F)$ is the time used by jobs assigned to machine i in the current F. The proof of the claim is by a standard inductive argument identical to the argument for the cost. Consider the step when the machine i is removed from M'. There can be at most two jobs assigned to machine i in later iterations and furthermore, fractionally machine i must be assigned at least one job. Hence, in either case, we have total time needed by all jobs assigned to machine i is at most $T'_i + T_i(F) \leq T_i + max_j p_{ij} \leq 2T_i$. This claim completes the proof of Theorem 7.1. ■

7.2 Multi-Criteria Spanning Trees

In this section, we give an approximation algorithm for the MULTI-CRITERIA SPANNING TREE problem. In this problem, we are given a graph $G = (V, E)$ and (non-negative) cost functions $c^0, c^1, \ldots, c^k$ on the edges and bounds $L_1, L_2, \ldots, L_k$ for the total edge cost of the tree under each of the cost functions c^i for each $1 \leq i \leq k$. The goal is to find a minimum c^0-cost tree which has c^i-cost at most L_i.

Ravi and Goemans [85] gave an algorithm for two cost functions c^0 and c^1 which, given a positive ϵ, returns a tree T with optimal c^0 cost and $c^1(T) \leq (1 + \epsilon)L_1$. The running time of the algorithm is polynomial for any fixed ϵ. We generalize their result and prove the following result.

Theorem 7.4 *Given a graph $G = (V, E)$ and cost functions $c^0, c^1, \ldots, c^k$ on the edges and bounds $L_1, L_2, \ldots, L_k$ for each of the cost function except c^0 and given any fixed $\epsilon > 0$, there exists a algorithm which returns a tree of optimal c^0-cost and has c^i-cost at most $(1 + \epsilon)L_i$. The running time of the algorithm is polynomial for fixed k and ϵ.*

7.2.1 Linear Program

We formulate the following linear programming relaxation for the problem which is a standard extension of the linear program for minimum spanning tree problem considered in Section 3.2.

$$\begin{array}{llrcll} \text{(LP-MCST)} & \text{minimize} & z_{LP} & = & \sum_{e\in E} c_e^0\, x_e & \\ & \text{subject to} & x(E(V)) & = & |V|-1, & \\ & & x(E(S)) & \leq & |S|-1, & \forall\, S\subset V \\ & & \sum_{e\in E} c_e^i x_e & \leq & L_i, & \forall\, 1\leq i\leq k \\ & & x_e & \geq & 0 & \forall\, e\in E \end{array}$$

7.2.2 Characterization of Vertex Solutions

We now give a characterization of any vertex solution of the linear program (LP-MCST). This follows directly from the Rank Lemma and the characterization of vertex solutions of the spanning tree linear program $LP_{ST}(G)$ in Section 3.2 (Lemma 3.13).

Lemma 7.5 *Let x be a vertex solution of the linear program (LP-MCST) such that $x_e > 0$ for each edge e and let $\mathcal{F} = \{S \subseteq V : x(E(S)) = |S| - 1\}$ be the set of all tight subset constraints. Then there exists a laminar family $\mathcal{L} \subseteq \mathcal{F}$ and $J = \{1 \leq i \leq k : \sum_{e\in E} c_e^i x_e = L_i\}$ a subset of tight cost constraints such that*

1. *The vectors $\{\chi(E(S)) : S \in \mathcal{L}\}$ are linearly independent.*
2. *span($\mathcal{L}$)=span($\mathcal{F}$).*
3. *$|\mathcal{L}| + |J| = |E|$.*

7.2.3 Iterative Algorithm

The algorithm proceeds in two phases. The first phase is the pruning step which we describe below. Observe that no feasible solution can include an edge whose c^i-cost is more than L_i. We extend this step further and *guess* all edges in the solution whose c^i-cost is at most $\frac{\epsilon}{k}L_i$. For any i there can be at most $\frac{k}{\epsilon}$ such edges in the optimal solution. Hence, trying all such possibilities for inclusion in a partial initial solution takes time $m^{\frac{k}{\epsilon}}$ where m is

Algorithm for Multi-Criteria Spanning Tree

1. Guess all edges in the optimal solution such that $c^i_e \geq \frac{\epsilon}{k} L_i$. Include all such edges in the solution and contract all such edges. Delete all other edges with $c^i_e \geq \frac{\epsilon}{k} L_i$ from G. Update L_i.
2. Find a vertex solution x and remove every edge e with $x_e = 0$.
3. Pick any minimum c^0-cost tree in the support graph.

Figure 7.3: Algorithm for Multi-criteria Spanning Trees.

the number of edges in G. There are k cost function to try which amounts to the total number of choices being at most $O(m^{\frac{k^2}{\epsilon}})$. After guessing these edges correctly, we throw away all other edges which have c^i cost more than ϵL_i and contract the guessed edges in the optimal solution. Clearly, the rest of the edges in the optimal solution form a spanning tree in the contracted graph. Also, now we have an instance where $c^i_e \leq \epsilon L_i$ for each e and i. We also update the bound L_i by subtracting the costs of the selected edges. Let L'_i denote the residual bounds. We solve the linear program (LP-MCST) with updated bounds L'_i. The algorithm is given in Figure 7.3.

Now we prove Theorem 7.4.

Proof of Theorem 7.4: We first prove the following simple claim which follows from Lemma 7.5.

Claim 7.6 *The support of $(LP-MCST)$ on a graph with n vertices has at most $n+k-1$ edges.*

Proof: From Lemma 7.5, we have $|E| = |\mathcal{L}| + |J|$. But $|\mathcal{L}| \leq n-1$ since $\mathcal{L}$ is a laminar family without singletons (see Proposition 2.10) and $|J| \leq k$ proving the claim. □

Observe that the c^0-cost of the tree is at most the cost of the LP-solution and hence is optimal for the correct guess of *heavy* edges. Now, we show that the c^i-cost is at most $L'_i + \epsilon L_i$. Observe that any tree must contain $n-1$ edges out of the $n+k-1$ edges in the support. Hence, the costliest c^i-cost tree costs no more than $k \cdot \frac{\epsilon}{k} L'_i = \epsilon L'_i$ more than the minimum c^i-cost tree. But that the fractional solution picks $n-1$ edges whose c^i-cost is at most L'_i. Thus the cost of the minimum c^i-cost tree is at most L'_i and the fractional solution which is at most L'_i and the cost of the costliest c_i-cost tree in the support costs no more than $L'_i + \epsilon L_i$. Adding the cost of edges guessed in the first step we obtain that

$$
\begin{array}{llrcll}
\text{(LP-MCMB)} & \text{minimize} & z_{LP} & = & \displaystyle\sum_{e \in V} c_e^0 \, x_e & \\
& \text{subject to} & x(V) & = & r(V), & \\
& & x(S) & \leq & r(S), & \forall\, S \subset V \\
& & \displaystyle\sum_{e \in V} c_e^i x_e & \leq & L_i, & \forall\, 1 \leq i \leq k \\
& & x_e & \geq & 0 & \forall\, e \in V
\end{array}
$$

Figure 7.4: Linear Program for Multi-Criteria Matroid Basis Problem.

the tree returned by the algorithm costs at most $L'_i + \epsilon L_i + L_i - L'_i = (1+\epsilon)L_i$. ■

7.3 Multi-Criteria Matroid Basis

In this section we generalize the result of Section 7.2 to MULTI-CRITERIA MATROID BASIS problem. In an instance of the MULTI-CRITERIA MATROID BASIS problem, we are given a matroid $M = (V, \mathcal{I})$, cost functions $c^i : V \to \mathbb{R}_+$ for $0 \leq i \leq k$, bounds L_i for each $1 \leq i \leq k$ and the task is to the find the minimum c^0-cost basis of M such that c^i-cost is at most L_i. When M is the graphic matroid, this problem reduces to the MULTI-CRITERIA SPANNING TREE problem which we considered in the previous section. We prove the following theorem.

Theorem 7.7 *Given any $\epsilon > 0$ there exists a polynomial time algorithm which given an instance of the* MULTI-CRITERIA MATROID *problem returns a basis B of M of optimal c^0-cost and $c^i(B) \leq (1+\epsilon)L_i$ for each $1 \leq i \leq k$. The running time of the algorithm is polynomial for fixed ϵ and k.*

We prove Theorem 7.7 using the iterative method.

7.3.1 Linear Programming Relaxation

We now formulate the following linear programming relaxation (LP-MCMB) for the problem which is a straightforward extension of the linear program for the MINIMUM COST MATROID BASIS problem considered in Section 3.4. Here r denotes the rank function of the matroid M.

The polynomial time solvability of the linear program (LP-MCMB) follows from polynomial time separation for the linear program $LP_{mat}(M)$ as discussed in Section 3.4.

7.3.2 Characterization of Vertex Solutions

The following characterization is immediate from Lemma 3.31.

Lemma 7.8 *Let x be a vertex solution of the linear program (LP-MCMB) such that $x_e > 0$ for each $e \in V$ and let $\mathcal{F} = \{S \subseteq V : x(S) = r(S)\}$ be the set of all tight subset constraints. Then there exists a chain $\mathcal{C} \subseteq \mathcal{F}$ and $J = \{1 \leq j \leq k : \sum_{e \in V} c_e^i x_e = L_i\}$ a subset of tight cost constraints such that*

1. *The vectors $\{\chi(S) : S \in \mathcal{C}\}$ are linearly independent.*
2. *span($\mathcal{C}$)=span($\mathcal{F}$).*
3. *$|\mathcal{C}| + |J| = |V|$.*

7.3.3 Iterative Algorithm

The algorithm generalizes the algorithm given in Section 7.2.3 for the MULTI-CRITERIA SPANNING TREE PROBLEM. We first perform a pruning step to *guess* all elements in the optimal solution with c^i-cost at most $\frac{\epsilon L_i}{k}$ for any $1 \leq i \leq k$. Then we solve the linear program (LP-MCMB) for the residual problem and remove all elements which the liner program sets to a value of zero. We then select the minimum cost basis under cost function c^0 ignoring the rest of the cost functions.

Now we prove Theorem 7.7.

Proof of Theorem 7.7: We first prove the following simple claim which follows from Lemma 7.8.

Claim 7.9 *The support to the $LP_{matroid}$ on a matroid with $r(V) = n$ has at most $n + k$ elements.*

Proof: From Lemma 7.8, we have $|V| = |\mathcal{C}| + |J|$. But $|\mathcal{C}| \leq r(V)$ since $\mathcal{C}$ is a chain and $x(C)$ equal a distinct integer between 1 and $r(V)$ for each $C \in \mathcal{C}$. Also $|J| \leq k$ proving the claim. □

Algorithm for Multi-Criteria Matroid Basis

1. Guess all elements in the optimal solution such that $c_e^i \geq \frac{\epsilon}{k} L_i$.
   - Include all such elements in the solution and update the matroid by contracting these elements (see Definition 3.28).
   - Delete all other heavy elements e with $c_e^i \geq \frac{\epsilon}{k} L_i$ for any i from M (see Definition 3.27).
   - Update L_i.
2. Find a vertex solution x of (LP-MCMB) for the residual problem and remove every element e with $x_e = 0$.
3. Pick any minimum c^0-cost basis in the support.

Figure 7.5: Algorithm for Multi-criteria Matroid Basis.

Observe that the c^0-cost of the basis returned is at most the cost of the LP-solution and hence is optimal. Now, we show that the c^i-cost is at most $L_i' + \epsilon L_i$. Observe that any basis must contain $r(V)$ elements out of the $r(V)+k$ elements in the support. Hence, the costliest c^i-cost basis differs from the minimum c^i-cost basis by at most $k \cdot \frac{\epsilon}{k} L_i' = \epsilon L_i'$. But the minimum c^i-cost basis has c^i-cost at most the cost of fractional basis L_i' thus proving Theorem 7.7. ■

7.4 Degree Constrained Matroids

We consider the Minimum Bounded-Degree Matroid Basis problem, which is a generalization of the Minimum Bounded-Degree Spanning Tree problem. We are given a matroid $M = (V, \mathcal{I})$, a cost function $c : V \to \mathbb{R}$, a hypergraph $H = (V, E)$, and lower and upper bounds $f(e)$ and $g(e)$ for each hyperedge $e \in E(H)$. The task is to find a basis B of minimum cost such that $f(e) \leq |B \cap e| \leq g(e)$ for each hyperedge $e \in E(H)$. One motivation for considering the matroid generalization was the following problem posed by Frieze [36]: "Given a binary matroid M_A over the columns of a $0,1$-matrix A and bounds g_i for each row i of A, find a basis B of matroid M_A such that there are at most g_i ones in any row among columns in B". Our main result is the following:

Theorem 7.10 *There exists a polynomial time algorithm for the* Minimum Bounded-Degree Matroid Basis *problem which returns a basis B of cost at most* opt *such that $f(e) - 2\Delta + 1 \leq |B \cap e| \leq g(e) + 2\Delta - 1$ for each $e \in E(H)$. Here $\Delta = \max_{v \in V} |\{e \in$*

$E(H) : v \in e\}|$ is the maximum degree of the hypergraph H and OPT *is the cost of an optimal solution which satisfies all the hyperedge intersection bounds.*

This theorem can be improved if only upper bounds (or only lower bounds) are present. The proof of the improvement uses the proof technique of Bansal et al. [5], who worked independently on the MINIMUM CROSSING SPANNING TREE problem and obtained the following result for that special case.

Theorem 7.11 *There exists a polynomial time algorithm for the* MINIMUM BOUNDED-DEGREE MATROID BASIS *problem with only upper bounds which returns a basis B of cost at most* OPT *such that $|B \cap e| \leq g(e) + \Delta - 1$ for each $e \in E(H)$. An analogous result holds when only lower bounds are present.*

It should be noted that this does not match the result in Section 4.3 on minimum bounded-degree spanning trees, since that result violates the degree bounds by at most 1 even when both upper and lower bounds are present.

First we show some applications of the main results. Then we present the proofs of the main results.

Applications

In this section we highlight some applications of the main results.

Minimum Crossing Spanning Tree

In the MINIMUM CROSSING SPANNING TREE problem, we are given a graph $G = (V, E)$ with edge cost function c, a collection of cuts (edge subsets) $\mathcal{C} = \{C_1, \ldots, C_m\}$ and bound g_i for each cut C_i. The task is to find a tree T of minimum cost such that T contains at most g_i edges from cut C_i where $C_i = \delta(S_i)$ for some $S_i \subset V$. See [7] for various applications of this problem. The MINIMUM BOUNDED-DEGREE SPANNING TREE problem is the special case where $\mathcal{C} = \{\delta(v) : v \in V\}$. The following result (see also [5]) can be obtained as a corollary of Theorem 7.11. Note that $d = 2$ for the MINIMUM BOUNDED-DEGREE SPANNING TREE problem.

Corollary 7.12 *[5] There exists a polynomial time algorithm for the* MINIMUM CROSSING SPANNING TREE *problem that returns a tree T with cost at most* OPT *and such that T contains at most $g_i + d - 1$ edges from cut C_i for each i where $d = \max_{e \in E} |\{C_i : e \in C_i\}|$. Here* OPT *is the cost of an optimal solution which satisfies all the cut intersection bounds.*

Proof: Let $M = (E, \mathcal{I})$ denote the graphic matroid over the graph G. The hypergraph H is defined with $V(H) = E(G)$ and $E(H) = \{C_i : 1 \leq i \leq m\}$. Note that $\Delta = \max_{v \in V(H)} |\{e \in E(H) : v \in e\}| = \max_{e \in E(G)} |\{C_i : e \in C_i\}| = d$. So, using Theorem 7.11, we obtain a basis T of matroid M (which is a spanning tree), such that $|T \cap C_i| \leq g_i + d - 1$. □

Minimum Bounded-Ones Binary Matroid Basis

For the Minimum Bounded-Ones Binary Matroid Basis problem posed by Frieze [36], we are given a binary matroid M_A over the columns of a 0,1-matrix A and bounds g_i for each row i of A. The task is to find a minimum cost basis B of matroid M_A such that there are at most g_i ones in any row among columns in B. The following result is obtained as a corollary of Theorem 7.11.

Corollary 7.13 *There exists a polynomial time algorithm for the* Minimum Bounded-Ones Binary Matroid Basis *problem which returns a basis B of cost at most* opt *such that there are at most $g_i + d - 1$ ones in any row restricted to columns of B. Here d is the maximum number of ones in any column of A and* opt *is the cost of an optimal solution satisfying all the row constraints.*

Proof: Let $M = M_A$ and define a hypergraph H where the vertex set is the columns of A. The hyperedges correspond to rows of A where $e_i = \{A^j : A_{ij} = 1\}$ where A^j is the j^{th} column of A. Note that $\Delta = \max_{v \in V(H)} |\{e \in E(H) : v \in e\}| = \max_j |\{i : a_{ij} = 1\}| = d$, which is the maximum number of ones in any column of A. So, using Theorem 7.11, we obtain a basis of $M = M_A$ such that number of ones in any row is at most $g_i + d - 1$. □

Minimum Bounded-Degree Spanning Tree Union

In the Minimum Bounded-Degree Spanning Tree Union problem, we are given a graph $G = (V, E)$ with edge cost function c, a positive integer k, and lower and upper degree bounds $f(v)$ and $g(v)$ for each vertex v. The task is to find a subgraph H which is the union of k edge-disjoint spanning trees and the degree of v in H is between $f(v)$ and $g(v)$. The Minimum Bounded-Degree Spanning Tree problem is a special case when $k = 1$. Theorem 7.11 implies the following result, which is optimal in terms of the degree upper bounds.

Corollary 7.14 *There exists a polynomial time algorithm for the* Minimum Bounded-Degree Spanning Tree Union *problem which returns a subgraph G of cost at most* opt

which is the union of k edge-disjoint spanning trees and the degree of v in H is at most $g(v)+1$. Here OPT *is the cost of an optimal solution which satisfies all the degree upper bounds given by the function g.*

Proof: Let $M=(E,\mathcal{I})$ denote the union of k graphic matroids over the graph G, which is a matroid by the matroid union theorem (see Chapter 43 in [91]). The hypergraph H is defined with $V(H)=E(G)$ and $E(H)=\{\delta(v) : v \in V(G)\}$. Note that $\Delta = \max_{v\in V(H)} |\{e \in E(H) : v \in e\}| = \max_{e\in E(G)} |\{\delta(v) : v \in V(G) \wedge e \in \delta(v)\}| = 2$. So, using Theorem 7.11, we obtain a basis T of matroid M (which is the union of k edge-disjoint spanning trees), such that $|T \cap C_i| \leq g_i + 1$. □

7.4.1 Linear Program

We now give the linear programming relaxation $LP_{mat}(M,H)$ for the MINIMUM BOUNDED-DEGREE MATROID BASIS problem. Let $r : 2^V \to \mathbb{Z}_+$ denote the rank function of matroid M.

$$\text{minimize} \quad c(x) = \sum_{v\in V} c_v\, x_v \tag{7.5}$$

$$\text{subject to} \quad x(V) = r(V) \tag{7.6}$$

$$x(S) \leq r(S) \qquad \forall\, S \subseteq V \tag{7.7}$$

$$f(e) \leq x(e) \leq g(e) \qquad \forall\, e \in E(H) \tag{7.8}$$

$$0 \leq x_v \leq 1 \qquad \forall\, v \in V \tag{7.9}$$

This linear program is exponential in size but can be separated over in polynomial time if given an access to the independent set oracle [20]. Given a matroid $M=(V,\mathcal{I})$ and an element $v \in V$, we denote by $M\backslash v$ the matroid obtained by deleting v, i.e., $M\backslash v = (V',\mathcal{I}')$ where $V' = V\backslash\{v\}$ and $\mathcal{I}' = \{S \in \mathcal{I} : v \notin S\}$. We also denote by M/v the matroid obtained by contracting v, i.e., $M/v = (V',\mathcal{I}')$ where $V' = V\setminus\{v\}$ and $I' = \{S\backslash\{v\} : S \in \mathcal{I}, v \in S\}$.

7.4.2 Characterization of vertex solutions

We have the following characterization of an extreme point of the linear program which follows directly from the Rank Lemma and Lemma 3.31.

1. Initialization $B \leftarrow \emptyset$,
2. While B is not a basis do
   (a) Find a vertex optimal solution x to $LP_{mat}(M, H)$. Delete v such that $x_v = 0$. Update each edge $e \in E(H)$ let $e \leftarrow e \setminus \{v\}$. Update Matroid $M \leftarrow M \setminus v$.
   (b) For each element v with $x_v = 1$, include v in B and decrease $f(e)$ and $g(e)$ by 1 for each $e \ni v$. Update $M \leftarrow M/v$.
   (c) For every $e \in E(H)$ such that $|e| \leq 2\Delta$, remove e from $E(H)$.
3. Return B.

Figure 7.6: Algorithm for the Minimum Bounded-Degree Matroid Basis.

Claim 7.15 *Let $\mathcal{T} = \{S \subseteq V : x(S) = r(S)\}$ be the collection of all tight sets at solution x. There is a set $E' \subseteq E$ of tight hyperedges and a chain $\mathcal{L} \subseteq \mathcal{T}$ such that*

1. *$\{\chi(S) : S \in \mathcal{L}\} \cup \{\chi(e) : e \in E'\}$ are linearly independent vectors*
2. *$span(\{\chi(S) : S \in \mathcal{L}\}) = span(\{\chi(S) : S \in \mathcal{T}\})$.*
3. *$|V| = |E'| + |\mathcal{L}|$.*

7.4.3 Iterative Algorithm

The algorithm is given in Figure 7.6. Suppose that the algorithm terminates successfully. Then Theorem 7.10 follows from a similar argument as in [96], which is sketched as follows. Firstly, observe that the matroid M is updated to $M \setminus v$ whenever we remove v such that $x_v = 0$ and updated to M/v whenever we pick v such that $x_v = 1$. A simple verification shows that the *residual* linear programming solution (current LP solution restricted to $V \setminus \{v\}$) remains a feasible solution for the modified linear program in the next iteration. In Step2c we remove a degree constraint hence, the current linear programming solution remains a feasible solution. Now, a simple inductive argument shows that by only picking elements with $x_v = 1$, the cost of the returned basis is no more than the cost of the original vertex optimal solution. Also, since we only remove a degree constraint of a hyperedge when it contains at most 2Δ elements, the degree constraints are violated by at most $2\Delta - 1$. Therefore, it remains to show that the algorithm always terminates successfully. That is, it can always find an element v with $x_v = 1$ in Step 2b or it can find a hyperedge e with $|e| \leq 2\Delta$ in Step 2c.

Suppose for contradiction neither of the above conditions hold. Hence, $0 < x_v < 1$ for each $v \in V$ and $|e| > 2\Delta$ for each $e \in E(H)$. From Lemma 7.15 there is a set $E' \subseteq E$ of tight hyperedges such that $\{\chi(S) : S \in \mathcal{L}\} \cup \{\chi(e) : e \in E'\}$ are linearly independent vectors and $|V| = |E'| + |\mathcal{L}|$. We now derive a contradiction to this by a counting argument. We assign 2Δ tokens to each vertex $v \in V$ for a total of $2\Delta|V|$ tokens. We then redistribute the tokens so that each hyperedge in E' collects at least 2Δ tokens, each member of $\mathcal{L}$ collects at least 2Δ tokens, and there are still at least one extra token. This implies that $2\Delta|V| > 2\Delta|E'| + 2\Delta|\mathcal{L}|$, which gives us the desired contradiction.

Proof of Theorem 7.10: The reassignment is as follows. Each element v gives Δ tokens to the smallest member in $\mathcal{L}$ it is contained in and one token to each edge $e \in E'$ it is present in. As any element is contained in at most Δ edges, thus the distribution is valid and we distribute at most 2Δ tokens per element. Now, consider any set $S \in \mathcal{L}$ and let R be the largest set in $\mathcal{L}$ contained in S. We have $x(S) = r(S)$ and $x(R) = r(R)$. Thus, we have $x(S \setminus R) = r(S) - r(R)$. As constraints for R and S are linearly independent and $x_v > 0$ for each $v \in V$, this implies $r(S) \neq r(R)$. Since r is a matroid rank function, $r(S) - r(R) \geq 1$ as they are both integers. Since $0 < x_v < 1$, this implies $|S \setminus R| \geq 2$. Thus, S can collect at least 2Δ tokens, Δ tokens from each element in $S \setminus R$, as required. Consider any hyperedge $e \in E'$. As $|e| \geq 2\Delta$ and it can collect one token from each element in e, there are at least 2Δ tokens for each edge e, as required.

Now, it remains to argue that there is an extra token left. If $V \notin \mathcal{L}$ or any of the elements is in strictly less than Δ hyperedges of E' then we have one extra token. Else, $\sum_{e \in E'} \chi(e) = \Delta \cdot \chi(V)$ which gives dependence among the constraints as $V \in \mathcal{L}$. Hence, we have the desired contradiction, and the proof of Theorem 7.10 follows. ■

Now we show how to use the proof technique of Bansal et al [5] to obtain Theorem 7.11.

Proof of Theorem 7.11: The proof for upper bounds is similar to the proof of Theorem 7.10 except for the counting argument. The only important difference is that we remove a hyperedge e if $g(e) + \Delta - 1 \geq |e|$; this is possible since in that case the degree upper bound on e can be violated by at most $\Delta - 1$. It follows that we may assume that $|e| - g(e) \geq \Delta$ for all hyperedges.

The proof that $|V| > |E'| + |\mathcal{L}|$ if $0 < x < 1$ goes as follows. Let $\mathcal{L} = \{S_1, \ldots, S_k\}$, where $S_1 \subsetneq S_2 \subsetneq \cdots \subsetneq S_k$, and let $S_0 := \emptyset$. Then $|e| - x(e) \geq \Delta$ for every $e \in E'$, and

$x(S_i \setminus S_{i-1}) = r(S_i) - r(S_{i-1}) \geq 1$ for $i = 1, \ldots, k$. Using these inequalities, we obtain that

$$|E'| + |\mathcal{L}| \leq \sum_{e \in E'} \frac{|e| - x(e)}{\Delta} + \sum_{i=1}^{k} x(S_i \setminus S_{i-1}) = \sum_{v \in V} \frac{1 - x(v)}{\Delta} |\{e \in E' : v \in e\}| + x(S_k) \leq |V|,$$

and if equality holds, then $|\{e \in E' : v \in e\}| = \Delta$ for every $v \in V$ and $S_k = V$. But then $\Delta \cdot \chi(S_k) = \sum_{e \in E'} \chi(e)$, which contradicts the linear independence of E' and $\mathcal{L}$.

If only lower bounds are present, then we can delete a hyperedge e if $f(e) \leq \Delta - 1$, so we may assume that $f(e) \geq \Delta$ for all hyperedges. To show $|V| > |E'| + |\mathcal{L}|$ we use that $x(e) \geq \Delta$ for every $e \in E'$ and $|S_i \setminus S_{i-1}| - x(S_i \setminus S_{i-1}) \geq 1$ for $i = 1, \ldots, k$, where the latter holds because $x(S_i \setminus S_{i-1}) < |S_i \setminus S_{i-1}|$ and both are integer. Thus

$$\begin{aligned} |E'| + |\mathcal{L}| &\leq \sum_{e \in E'} \frac{x(e)}{\Delta} + \sum_{i=1}^{k} (|S_i \setminus S_{i-1}| - x(S_i \setminus S_{i-1})) \\ &= \sum_{v \in V} \frac{x(v)}{\Delta} |\{e \in E' : v \in e\}| + |S_k| - x(S_k) \leq |V|, \end{aligned}$$

and the claim follows similarly as for upper bounds. ■

8

Conclusion

In this thesis, we showed that iterative techniques give a general methodology to deal with degree constraints in network design problems. The techniques enable us to obtain almost optimal results for a large class of degree constrained network design problems. In undirected graphs, we obtained first *additive* approximation algorithms for a large class of bounded-degree network design problems which violate the degree bounds by only a small additive amount. Moreover, the cost of the solution returned is close to optimal. Some of our main results that we obtained are the following.

- We give a polynomial time algorithm for the Minimum Bounded-Degree Spanning Tree problem which returns a tree of optimal cost and such that degree of any vertex v in the tree is at most $B_v + 1$.
- We obtain bi-criteria approximation algorithm for the Minimum Bounded-Degree Steiner Tree problem, Minimum Bounded-Degree Steiner Forest problem, Minimum Bounded-Degree Steiner Network problem. The solution returned by the algorithm costs at most twice the optimal solution and the degree of any vertex violates its degree bound by an additive error which depends on the maximum connectivity requirement.
- As a corollary to the previous results, we also obtain first additive approximation algorithms for Bounded-Degree Steiner Forest problem and Bounded-Degree k-edge connected subgraph problem for bounded k.
- We obtain constant factor bi-criteria approximation algorithm for the Minimum Bounded-Degree Arborescence problem where both the cost and the maximum degree of the solution is within constant multiplicative factor of the optimal solution.

- We use the iterative method for various other problems to obtain a polynomial time approximation scheme (PTAS) for the multi-criteria spanning tree problem and the multi-criteria matroid basis problem, 2-approximation for the generalized assignment problem and additive approximation algorithm for degree constrained matroid basis problem.

8.1 Further Work

In directed graphs, we gave constant factor bi-criteria approximation algorithms for degree constrained network design problems which violate both the cost and the degree by a constant multiplicative factor. Subsequently, using iterative methods, Bansal, Khandekar and Nagarajan [5] obtained an $(\frac{1}{\epsilon}, \frac{B_v}{1-\epsilon} + 4)$-approximation algorithm for the Minimum Bounded Degree Arborescence problem for $0 < \epsilon \leq \frac{1}{2}$. Moreover, they obtain the first additive approximation algorithm for the bounded-degree arborescence problem with degree violation at most 2. In order to obtain additive guarantees on the degree bounds, however, the cost of the solution becomes unbounded. They show that this cost-degree tradeoff in their result is actually best possible using the natural linear programming relaxation [5], which is an exact formulation when degree constraints are absent.

Lau and Király [57] used the iterative method to obtain additive approximation algorithms for the degree constrained submodular flow problem.

8.2 Future Direction

Iterative techniques are quite versatile and we showed its application to a wide variety of degree constrained network design problems and other problems including multi-criteria spanning tree, multi-criteria matroids, degree constrained matroids and the generalized assignment problem. A natural question is whether they will have other major applications. We highlight two problems where these techniques might be useful. The first problem is the Traveling Salesperson Problem. The TSP problem is essentially a network design problems with degree constraints and our results on the Minimum bounded degree spanning tree problem do lead to better polyhedral results about the TSP in both the symmetric and the asymmetric case. A connected graph is called a 1-tree if it contains exactly one cycle including a special vertex v. Held and Karp [47, 48] showed that any fractional solution to the well-known subtour elimination LP for the TSP problem can be written as a convex combination of 1-trees. The following theorem (a weaker version of it was shown

by Goemans [42]) follows using the ideas in Section 4.2.

Theorem 8.1 *Any fractional solution to the subtour elimination LP for the STSP problem can be expressed as a convex combination of* 1*-trees where the maximum degree of each* 1*-tree is at most three.*

A similar polyhedral result also follows for the asymmetric TSP problem where the fractional solution can be represented as a convex combination of weakly connected 1-tree each having a maximum in-degree and out-degree of three. Whether these results or further extensions using iterative methods can lead to better approximation algorithms for asymmetric TSP or symmetric TSP is open.

The second problem we state is the SINGLE SOURCE UNSPLITTABLE MIN-COST FLOW PROBLEM. In the SINGLE SOURCE UNSPLITTABLE MIN-COST FLOW problem, commodities must be routed simultaneously from a common source vertex to certain destination vertices in a given graph with edge capacities and costs; the demand of each commodity must be routed along a single path so that the total flow through any edge is at most its capacity. Moreover, the total cost must not exceed a given budget. Skutella [78] gives an algorithm which finds an unsplittable flow whose cost is bounded by the cost of the initial flow f and the flow value on any edge e is less than $2f(e) + d_{max}$ where f is a the minimum cost fractional flow satisfying all capacity constraints and d_{max} is the maximum demand. Goemans [78] conjectures that there is an unsplittable flow where the flow value on the edge is at most $f(e) + d_{max}$ and the flow is of optimal cost. Dinitz et al. [24] prove this in the version without costs. A special case for the unsplittable flow problem is the restricted assignment problem. The restricted assignment problem is also a special case of the generalized assignment problem considered in Section 7.1 where $p_{ij} \in \{p_j, \infty\}$ for each machine i and job j. The conjecture holds true for the restricted assignment problem and this also follows from the result on generalized assignment problem discussed in Section 7.1. Whether the iterative techniques can be used to prove the conjecture of Goemans [78] is open.

The introduction of the iterative rounding technique by Jain [53] led to its applications on various network design problems [14, 34, 38, 101]. Our work adds new relaxation ideas to iterative techniques and allows us to apply it to a larger set of problems. We believe that the iterative methods will be further extended and established as a general technique to achieve approximation algorithms for combinatorial optimization problems.

Bibliography

[1] Agrawal, P. Klein, and R. Ravi. When Trees Collide: An Approximation Algorithm for the Generalized Steiner Problem on Networks. In *Proceedings of the Twenty-Third Annual ACM symposium on Theory of Computing*, pages 134–144, 1991.

[2] K. Appel and W. Haken. Every Planar Map is Four Colorable. *Contemporary Math.*, 98, 1989.

[3] J. Aráoz and J. Green-Krotki W. Cunningham, J. Edmonds. Reductions to 1-Matching Polyhedra. *Networks*, 13:455–473, 1983.

[4] M. Balinski. Establishing the Matching Polytope. *Journal of Combinatorial Theory*, 13:1–13, 1972.

[5] N. Bansal, R. Khandekar, and V. Nagarajan. Additive Guarantees for Degree Bounded Directed Network Design. In *Proceedings of the Fourtieth Annual ACM Symposium on Theory of Computing (STOC)*, 2008.

[6] J. Beck and T. Fiala. "Integer-Making" Theorems. *Discrete Applied Mathematics*, 3:1–8, 1981.

[7] V. Bilo, V. Goyal, R. Ravi, and M. Singh. On the Crossing Spanning Tree Problem. In *Proceedings of 7th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)*, 2004.

[8] D. Birkhoff. Tres Observaciones Sobre el Algebra Lineal. *Universidad Nacional de Tucuman Revista , Serie A*, 5:147–151, 1946.

[9] F. Bock. An Algorithm to construct a Minimum Directed Spanning Tree in a Directed Network. *In Developments in Operations Research*, pages 29–44, 1971.

[10] Otakar Boruvka. O Jistém Problému Minimálním (About a Certain Minimal Problem) (in Czech, German summary). *Práce Mor. Prírodoved. Spol. v Brne III*, 3:37––58, 1926.

[11] Camion. Characterization of Totally Unimodular Matrices. In *Proceedings of the American Mathematical Society*, volume 16, pages 1068–1073, 1965.

[12] K. Chaudhuri, S. Rao, S. Riesenfeld, and K. Talwar. What would Edmonds do? Augmenting paths and witnesses for degree-bounded MSTs. In *Proceedings of International Workshop on Approximation Algorithms for Combinatorial Optimization Problems*, pages 26–39, 2005.

[13] K. Chaudhuri, S. Rao, S. Riesenfeld, and K. Talwar. Push Relabel and an Improved Approximation Algorithm for the Bounded-Degree MST Problem. In *Proceedings of International Colloquium on Automata, Languages and Programming*, 2006.

[14] J. Cheriyan, S. Vempala, and A. Vetta. Network Design via Iterative Rounding of Setpair Relaxations. *Combinatorica*, 26:255–275, 2006.

[15] Y. J. Chu and T. H. Liu. On the Shortest Arborescence of a Directed Graph. *Science Sinica*, 14:1396–1400, 1965.

[16] V. Chvátal. *Linear Programming*. Freeman, 1983.

[17] S. A. Cook. The Complexity of Theorem-Proving Procedures. In *Proceedings of the Third Annual ACM Symposium on theory of Computing*, pages 151–158, 1971.

[18] William J. Cook, William H. Cunningham, William R. Pulleyblank, and Alexander Schrijver. *Combinatorial Optimization*. John Wiley & Sons, Inc., New York, NY, USA, 1998.

[19] G. Cornuejols, J. Fonlupt, and D. Naddef. The Traveling Salesman Problem on a Graph and some related Integer Polyhedra. *Mathematical Programming*, 33:1–27, 1985.

[20] W. H. Cunningham. Testing Membership in Matroid Polyhedra. *Journal of Combinatorial Theory B*, 36:161–188, 1987.

[21] W.H. Cunningham and A.B. Marsh III. A Primal Algorithm for Optimum Matching. *Math. Programming Stud.*, 8:50–72, 1978.

[22] G. B. Dantzig. Comments on J. Von Neumann's "The Problem of Optimal Assignment on a Two-Person Game". Technical Report 435, RAND Corporation, Santa Monica, California, 1952.

[23] George B. Dantzig. Maximization of a Linear Function of Variables Subject to Linear Inequalities. In T.C. Koopmans, editor, *Activity Analysis of Production and Allocation*. Wiley, New York, 1951.

[24] Y. Dinitz, N. Garg, and M. Goemans. On the Single-Source Unsplittable Flow Problem. *Combinatorica*, 19:1–25, 1999.

[25] J. Edmonds. Maximum Matching and a Polyhedron with 0,1-Vertices. *J. Res. Natl. Bur. Stand*, 69:125–130, 1965.

[26] J. Edmonds. Paths, Trees and Flowers. *Canadian Journal of Mathematics*, 17:449–467, 1965.

[27] J. Edmonds. Optimum Branchings. *Journal of Res. National Bureau of Standards*, 71B:233–240, 1967.

[28] J. Edmonds. Submodular Functions, Matroids, and certain Polyhedra. In *Proceedings of the Calgary International Conference on Combinatorial Structures and their Application*, pages 69–87. Gordon and Breach, New York, 1970.

[29] J. Edmonds. Matroids and the Greedy Algorithm. *Mathematical Programming*, 1:125–136, 1971.

[30] J. Edmonds. Matroid Intersection. *Ann. Discrete Math*, 4:39–49, 1979.

[31] J. Edmonds and R. Giles. A Min-Max Relation for Submodular Functions on Graphs. *Annals of Discrete Mathematics*, 1:185––204, 1977.

[32] J. Egerváry. Matrixok kombinatorius tulajdonsagairol [in Hungarian: On combinatorial properties of matrices]. *Matematikaies Fizikai Lapok*, 38:16–28, 1931.

[33] T. Feder, R. Motwani, and A. Zhu. k-Connected Spanning Subgraphs of Low Degree. Technical Report 41, The Electronic Colloquium on Computational Complexity, 2006.

[34] L. Fleischer, K. Jain, and D.P. Williamson. Iterative Rounding 2-Approximation Algorithms for Minimum-Cost Vertex Connectivity Problems. *Journal of Computer and System Sciences*, 72, 2006.

[35] A. Frank. Kernel Systems of Directed Graphs. *Acta Sci Math*, 41:63–76, 1979.

[36] Alan Frieze. Personal communication, March 2007.

[37] M. Fürer and B. Raghavachari. Approximating the Minimum-Degree Steiner Tree to within One of Optimal. *Journal of Algorithms*, 17:409–423, 1994.

[38] Harold N. Gabow. On the Linfinity-Norm of Extreme Points for Crossing Supermodular Directed Network LPs. In *Proceedings of Integer Programming and Combinatorial Optimization*, pages 392–406, 2005.

[39] D. Gale. Optimal Assignments in an Ordered Set: an Application of Matroid Theory. *J. Combin. Theory*, 4:176–180, 1968.

[40] M. X. Goemans. Course Notes: Linear Programming. *http://ocw.mit.edu/OcwWeb/Electrical-Engineering-and-Computer-Science/6-854JFall2001/StudyMaterials/index.htm.*

[41] Michel X. Goemans and David P. Williamson. The Primal-Dual Method for Approximation Algorithms and its Application to Network Design Problems. In *Approximation Algorithms for NP-hard Problems.* PWS Publishing Co., Boston, MA, 1996.

[42] M.X. Goemans. Minimum Bounded-Degree Spanning Trees. In *Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science*, pages 273–282, 2006.

[43] M.X. Goemans and D.P. Williamson. A General Approximation Technique for Constrained Forest Problems. *SIAM Journal on Computing*, 24:296–317, 1995.

[44] R. L. Graham, M. Grötschel, and L. Lovász, editors. *Handbook of Combinatorics (Vol. 2).* MIT Press, Cambridge, MA, USA, 1995.

[45] M. Grötschel, L. Lovász, and A. Schrijver. The Ellipsoid Method and its Consequences in Combinatorial Optimization. *Combinatorica*, 1:169–197, 1981.

[46] J. M. Hammersley and J. G. Mauldon. General Principles of Antithetic Variates. *Proc. Cambridge Philosophical Society*, 52:476–481, 1956.

[47] M. Held and R.M. Karp. The Traveling-Salesman Problem and Minimum Spanning Trees. *Operations Research*, 18:1138––1162, 1970.

[48] M. Held and R.M. Karp. The Traveling-Salesman Problem and Minimum Spanning Trees: Part II. *Mathematical Programming*, 1:6—-25, 1971.

[49] D.S. Hochbaum, editor. *Approximation Algorithms for NP-hard Problems.* PWS Publishing Co., Boston, MA, USA, 1997.

[50] Hoffman and Wielandt. The Variation of the Spectrum of a Normal Matrix. *Journal of Duke Math.*, 1953.

[51] A. J. Hoffman and R. Oppenheim. Local Unimodularity in the Matching Polytope. *Annals of Discrete Mathematics*, 2:201–209, 1978.

[52] K. M. Hoffman and R. Kunze. *Linear Algebra.* Prentice Hall, 1971.

[53] K. Jain. A Factor 2 Approximation Algorithm for the Generalized Steiner Network Problem. *Combinatorica*, 21:39–60, 2001.

[54] N. Karmarkar. A New Polynomial-Time Algorithm for Linear Programming. *Combinatorica*, 4:373–395, 1984.

[55] R. Karp. Reducibility Among Combinatorial Problems. *Complexity of Computer Computations*, pages 85–103, 1972.

[56] L. G. Khachian. A Polynomial Algorithm in Linear Programming. *Dokl. Akad. Nauk SSSR*, 244:1093–1096, 1979. English translation in Soviet Math. Dokl. 20, 191-194, 1979.

[57] Tamás Király and Lap Lau Chi. Degree constrained submodular flows. Technical Report TR-2007-09, Egerváry Research Group, Budapest, 2007. `www.cs.elte.hu/egres`.

[58] Tamas Király, Lap Chi Lau, and Mohit Singh. Degree Bounded Matroids and Submodular Flows. In *13th Conference on Integer Programming and Combinatorial Optimization*, 2008.

[59] P. Klein, R. Krishan, B. Raghavachari, and R. Ravi. Approximation Algorithms for finding Low-Degree Subgraphs. *Networks*, 44:203–215, 2004.

[60] J. Könemann and R. Ravi. A Matter of Degree: Improved Approximation Algorithms for Degree Bounded Minimum Spanning Trees. *SIAM Journal on Computing*, 31:1783–1793, 2002.

[61] J. Könemann and R. Ravi. Quasi-Polynomial Time Approximation Algorithm for Low-Degree Minimum-Cost Steiner Trees. In *Proceedings of the 23rd Conference on Foundations of Software Technology and Theoretical Computer Science*, 2003.

[62] J. Könemann and R. Ravi. Primal-Dual meets Local Search: Approximating MSTs with Nonuniform Degree Bounds. *SIAM Journal on Computing*, 34:763–773, 2005.

[63] D. König. Gráfok és mátrixok. *Matematikai és Fizikai Lapok*, 38:116—-119, 1931.

[64] T. C. Koopmans and M. J. Beckman. Assignment Problems in the Location of Economic Activities. *Econometrica*, 25:53–76, 1957.

[65] G. Kortsarz and Z. Nutov. Approximating Min-Cost Connectivity Problems. In T.F. Gonzalez, editor, *Approximation Algorithms and Metaheuristics*, 2006.

[66] Joseph. B. Kruskal. On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem. In *Proceedings of the American Mathematical Society*, volume 7, pages 48—50, 1956.

[67] H.W. Kuhn. The Hungarian Method for the Assignment Problem. *Naval Research Logistic Quarterly*, 2:83–97, 1955.

[68] L. C. Lau, R. Ravi, and M. Singh. *Iterative Rounding and Relaxation*. April 2008.

[69] L.C. Lau, S. Naor, M. Salavatipour, and M. Singh. Survivable Network Design with Degree or Order Constraints. In *40th ACM Symposium on Theory of Computing*, pages 651–660, 2007.

[70] L.C. Lau and M. Singh. Additive Approximation for Bounded Degree Survivable Network. In *40th ACM Symposium on Theory of Computing*, 2008.

[71] E. Lawler. *Combinatorial Optimization: Networks and Matroids*. Holt, Rinehart and Winston, New York, 1976.

[72] L. Lovász and M. Plummer. Matching theory. 1986.

[73] N. Maculan. A New Linear Programming Problem for the Shortest s-Directed Spanning Tree Problem. *Journal of Combinatorics, Information and System Science*, 11:53–56, 1986.

[74] T.L. Magnanti and L.A. Wolsey. Optimal trees. *Handbooks in operations research and management science: Network models*, pages 503–616, 1995.

[75] K. Menger. Zur allgemeinen kurventheorie. *Fund. Math.*, 10:96–115, 1927.

[76] L. Mirsky. Proofs of Two Theorems on Doubly Stochastic Matrices. *Proceedings of Amerecan Mathematical Society*, 9:371–374, 1958.

[77] L. Mirsky. Results and Problems in the Theory of Doubly-Stochastic Matrices. *Z. Wahrscheinlichkeitstheorie verw. Gebiete*, 1:319–334, 1963.

[78] M.Skutella. Approximating the Single-Source Unsplittable Min-Cost Flow Problem. *Mathematical Programming B*, 91:493–514, 2002.

[79] T. Nakasawa. Zur Axiomatik der Linearen Abhängigkeit. *I. Sci. Rep. Tokyo Bunrika Daigaku*, 1935.

[80] George L. Nemhauser and Laurence A. Wolsey. *Integer and Combinatorial Optimization.* Wiley-Interscience, 1999.

[81] J. Oxley. *Matroid Theory.* Oxford University Press, New York, 1992.

[82] M. W. Padberg and L. A. Wolsey. Fractional Covers for Forests and Matchings. *Mathematical Programming*, 29:1–14, 1984.

[83] R. C. Prim. Shortest Connection Networks and some Generalisations. *Bell System Technical Journal*, 36:1389-–1401, 1957.

[84] R. Rado. Note on Independence Functions. *Proc. London Math. Society*, 7:300–320, 1957.

[85] R. Ravi and M.X. Goemans. The Constrained Minimum Spanning Tree Problem. In *Fifth Scandinavian Workshop on Algorithm Theory, LNCS*, volume 1097, pages 66–75, 1996.

[86] R. Ravi, M.V. Marathe, S.S. Ravi, D.J. Rosenkrants, and H.B. Hunt. Approximation Algorithms for Degree-Constrained Minimum-Cost Network-Design Problems. *Algorithmica*, 31:58–78, 2001.

[87] R. Ravi, B. Raghavachari, and P. Klein. Approximation through Local Optimality: Designing Networks with Small Degree. In *Proceedings of the 12 Conference on Foundations of Sofware Tech. and Theoret. Comput. Sci.*, volume 652, pages 279–290, 1992.

[88] R. Ravi and M. Singh. Delegate and Conquer: An LP-based Approximation Algorithm for Minimum Degree MSTs. In *Proceedings of International Colloquium on Automata, Languages and Programming*, 2006.

[89] R. Ravi and M. Singh. Iterative Algorithms for Multi-Criteria Problems. *In Preparation*, 2008.

[90] G. Robins and A. Zelikovsky. Improved Steiner Tree Approximation in Graphs. In *Proceedings 10th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 770–779, 2000.

[91] A. Schrijver. *Combinatorial Optimization - Polyhedra and Efficiency.* Springer-Verlag, New York, 2005.

[92] Alexander Schrijver. Short Proofs on the Matching Polyhedron. *J. Comb. Theory, Ser. B*, 34:104–108, 1983.

[93] Alexander Schrijver. *Theory of Linear and Integer Programming.* John Wiley & Sons, Inc., New York, NY, USA, 1986.

[94] Alexander Schrijver. On the history of combinatorial optimization (till 1960). In K. Aardal, G.L. Nemhauser, and R. Weismantel, editors, *Handbook of Discrete Optimization*, pages 1–68. Elsevier, Amsterdam, July 24 2005.

[95] D. Shmoys and E. Tardos. An Approximation Algorithm for the Generalized Assignment Problem. *Mathematical Programming*, 62:461–474, 1993.

[96] M. Singh and L.C. Lau. Approximating Minimum Bounded Degree Spanning Trees to within One of Optimal. In *Proceedings of ACM-Symposium on Theory of Computing*, pages 661–670, 2007.

[97] E. Steinitz. *Algebraische Theorie der Körper*, 137:167–309, 1910.

[98] C. B. Tompkins. Machine Attacks on Problems whose Variables are Permutations. *Numerical Analysis (Proceedings of Symposia in Applied Mathematics)*, 6, 1956.

[99] V. Vazirani. *Approximation Algorithms.* Springer, 2001.

[100] V. Vizing. Critical Graphs with Given Chromatic Class. *Metody Diskret. Analiz.*, 5:9—-17, 1965.

[101] V.Melkonian and E.Tardos. Algorithms for a Network Design Problem with Crossing Supermodular Demands. *Networks*, 43:256–265, 2004.

[102] J. von Neumann. The Problem of Optimal Assignment Problem and a Certain 2-Person Game. *Unpublished Manuscript*, 1951.

[103] J. von Neumann. A Certain Zero-Sum Two-Person Game Equivalent to the Optimal Assignment Problem. In H. W. Kuhn and A. W. Tucker, editors, *Contributions to the Theory of Games*, pages 5–12. Princeton University Press, 1953.

[104] B. L. Van Der Waerden. Moderne Algebra Vol. I. Second Edition. 1937.

[105] H. Whitney. On the Abstract Properties of Linear Dependence. *American Journal of Mathematics*, pages 509–533, 1935.

[106] S. Win. On a Connection between the Existence of k-Trees and the Toughness of a Graph. *Graphs and Combinatorics*, 5:201–205, 1989.

[107] R. T. Wong. A Dual Ascent Approach for Steiner Tree Problems on a Directed Graph. *Mathematical Programming*, 28:271–287, 1984.

[108] S. J. Wright. *Primal-Dual Interior-Point Methods.* SIAM Publications, 1997.