# Iterative Methods in Combinatorial Optimization

Lap-Chi Lau
*Chinese University of Hong Kong*

R. Ravi
*Carnegie Mellon University*

Mohit Singh
*McGill University*

© Draft date March 1, 2010

# Contents

# Preface

**Audience:** As teachers and students of Combinatorial Optimization, we have often looked for good teaching material that illustrates the elegance of the classical results on matchings, trees, matroids and flows while at the same time trying to concentrate on methods that have seen continued application. With the advent of approximation algorithms, some of these techniques from exact optimization such as the primal-dual method have indeed proven their staying power. In this book, we describe what we believe is a simple and powerful method of the same ilk that is iterative in essence.

The core of the iterative methods we describe relies on a fundamental result in linear algebra that the row rank and column rank of a real matrix are equal. This seemingly elementary fact allows us via a counting argument to provide an alternate proof of the above-mentioned classical results; the method is constructive and the resulting algorithms are iterative with the correctness proven by induction. What's more, these methods generalize to accommodate a variety of additional constraints on these classical problems that render them NP-hard, but a careful adaptation of the iterative method leads to very effective approximation algorithms.

Our goal in this book has been to highlight the commonality and uses of this method and convince the readers of the generality and potential for future applications. We have used an elementary presentation style that should be accessible to anyone with introductory college mathematics exposure in linear algebra and basic graph theory. Whatever advanced material in these areas we require, we develop from scratch along the way. Some basic background on approximation algorithms such as is provided in the various books and surveys available on this subject will be useful in appreciating the power of the results we prove in this area, but other than the basic definition of an approximation algorithm and the understanding of polynomial-time complexity, no further technical background is required from this typically more advanced subject.

An important secondary goal of the book is to provide a framework and material for introductory courses in combinatorial optimization at the upper-class undergraduate and beginning graduate levels. We hope the common approach across the chapters gives a

comprehensive way to introduce these topics for the first time. The more advanced applications are useful illustrations for graduate students of their potential for future application in their research.

**History:** This book is inspired by the application of the iterative method in the field of approximation algorithms and its recent adaptations to prove performance guarantees for problems with two objectives. This adaptation showed us how the proof technique can be used to re-prove several classical results in combinatorial optimization and also in approximation algorithms in a unified way. The book owes its origin to the paper by Jain [78] describing a 2-approximation algorithm for a large class of minimum-cost network-design problems in undirected networks. While there are other earlier illustrations of the method in the literature, it is Jain's work that inspired the adaptation that led to the results in this monograph.

Jain's result itself was a breakthrough when it appeared, and demonstrated the power of his *iterative rounding* method to prove this result that was conjectured based on a long line of earlier papers that applied a different primal-dual method to these problems. In this sense, his method was a purely primal attack on the problem. His method was extended by Lau, Naor, Salavatipour and Singh [95] to degree-bounded network design problems. The adaptation of this method by Singh and Lau [134] to the degree-bounded minimum-cost spanning tree problem surprisingly involves no rounding at all! Instead, variables whose value are set to one in the linear programming relaxation are selected and the program is modified carefully to continue to yield this property. This explains the title of this monograph and also hints at how this adaptation now allows one to prove *exact* results since we no longer have to round any variables and lose optimality.

**Acknowledgments:** We are grateful to the many organizations whose support have enabled this work: National Science Foundation, Research Grants Council of Hong Kong, Microsoft Research, Kyoto University RIMS, Qatar Foundation, Carnegie Mellon University - Pittsburgh and Doha and the Chinese University of Hong Kong. We are also grateful to our families for their support of this endeavor. We hope you will enjoy reading this monograph as much as we did writing it.

# 1
# Introduction

In this first chapter we motivate our method via the assignment problem. Through this problem we highlight the basic ingredients and ideas of the method. We then give an outline of how a typical chapter in the rest of the book is structured, and how the remaining chapters are organized.

## 1.1 The Assignment Problem

Consider the classical assignment problem: Given a bipartite graph $G = (V_1 \cup V_2, E)$ with $|V_1| = |V_2|$ and weight function $w : E \to \mathbb{R}_+$, the objective is to match every vertex in $V_1$ with a distinct vertex in $V_2$ to minimize the total weight (cost) of the matching. This is also called the minimum weight bipartite perfect matching problem in the literature, and is a fundamental problem in combinatorial optimization. See Figure 1.1 for an example of a perfect matching in a bipartite graph.

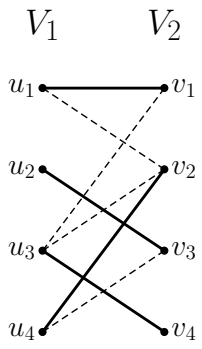


Fig. 1.1. The solid edges form a perfect matching in the bipartite graph.

One approach to the assignment problem is to model it as a linear programming problem. A linear program is a mathematical formulation of the problem with a system

of linear constraints which can contain both equalities and inequalities, and also a linear objective function that is to be maximized or minimized. In the assignment problem, we associate a *variable* $x_{uv}$ for every $(u, v) \in E$. Ideally, we would like the variables to take one of two values, zero or one (hence in the ideal case, they are binary variables). When $x_{uv}$ is set to one, we intend the model to signal that this pair is matched; when $x_{uv}$ is set to zero, we intend the model to signal that this pair is not matched. The following is a linear programming formulation of the assignment problem.

$$
\begin{array}{lll}
\text{minimize} & \displaystyle\sum_{u,v} w_{uv}\, x_{uv} & \\
\text{subject to} & \displaystyle\sum_{v:\{u,v\}\in E} x_{uv} = 1 & \forall\, u \in V_1 \\
& \displaystyle\sum_{u:\{u,v\}\in E} x_{uv} = 1 & \forall\, v \in V_2 \\
& x_{uv} \geq 0 & \forall\, \{u, v\} \in E
\end{array}
$$

The objective function is to minimize the total weight of the matching, while the two sets of linear equalities ensure that every vertex in $V_1$ is matched to exactly one vertex in $V_2$ in the assignment and vice-versa.

A fundamental result in the Operations Research literature [73] is the polynomial time solvability (as well as the practical tractability) of linear programming problems. There is also a rich theory of optimality (and certificates for it) that has been developed (see e.g., the text by Chvatal [29]). Using these results, we can solve the problem we have formulated above quite effectively for even very large problem sizes.

Returning to the formulation however, our goal is to find a "binary" assignment of vertices in $V_1$ to vertices in $V_2$, but in the solution returned, the $x$-variables may take fractional values. Nevertheless, for the assignment problem, a celebrated result that is a cornerstone of combinatorial optimization [30] states that for any set of weights that permit a finite optimal solution, there is always an optimal solution to the above LP (linear program) that takes binary values in all the $x$-variables.

Such *integrality* results of LPs are few and far between, but reveal rich underlying structure for efficient optimization over the large combinatorial solution space [130]. They have been shown using special properties of the constraint matrix of the problem (such as total unimodularity), or of the whole linear system including the right hand side (such as total dual integrality). This book is about a simple and fairly intuitive method that is able to re-prove many (but not all) of the results obtained by these powerful methods. One advantage of our approach is that it can be used to incorporate additional constraints that make the problem computationally hard, and allow us to derive good approximation algorithms with provable performance guarantee for the constrained versions.

### 1.2 Iterative Algorithm

Our method is iterative. Using the following two steps, it works inductively to show that the LP has an integral optimal solution.

- If any $x_{uv}$ is set to 1 in an optimal solution to the LP, then we take this pair as matched in our solution, and delete them both to get a smaller problem, and proceed to the next iteration.
- If any variable $x_{uv}$ is set to 0 in an optimal solution, we remove the edge $(u, v)$ to again get a smaller problem (since the number of edges reduces by 1) and proceed to the next iteration.

We continue the above iterations till all variables have been fixed to either 0 or 1. Given the above iterative algorithm, there are two claims that need to be proven. Firstly, that the algorithm works correctly, i.e., it can always find a variable with value 0 or 1 in each iteration and secondly, the matching selected is an optimal (minimum weight) matching. Assuming the first claim, the second claim can be proved by a simple inductive argument. The crux of the argument is that in each iteration our solution pays exactly what the fractional optimal solution pays. Moreover, the fractional optimal solution when restricted to the residual graph remains feasible for the residual problem. This allows us to apply an inductive argument to show that the matching we construct has the same weight as the fractional optimal solution, and is thus optimal. For the first claim, it is not clear a-priori that one can always find a variable with value 1 or 0 at every step. Indeed the example in Figure 1.2 shows that there might not be such a variable at some fractional optimal solution. However, we use the important concept of the extreme point (or vertex) solutions of linear program to show that the above iterative algorithm works correctly.

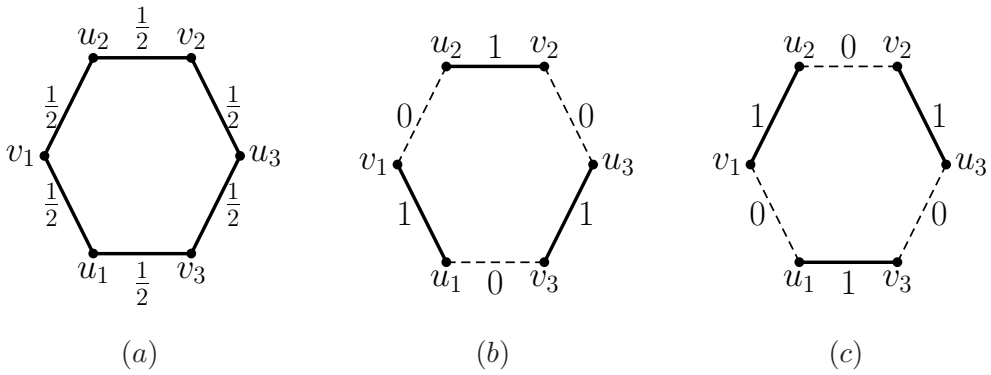


Fig. 1.2. In Figure (a), the fractional solution which places $\frac{1}{2}$ on all the edges is an optimal fractional solution but not an extreme point solution. The fractional solution in Figure (a) is the convex combination of the integral solutions in Figure (b) and Figure (c).

**Definition 1.2.1** *Let $P = \{x : Ax = b, x \geq 0\} \subseteq \mathbb{R}^n$. Then $x \in \mathbb{R}^n$ is an* **extreme point solution** *of $P$ if there does not exist a non-zero vector $y \in \mathbb{R}^n$ such that $x + y, x - y \in P$.*

Extreme point solutions are also known as vertex solutions and are equivalent to basic feasible solutions. These concepts are defined in Chapter 2. Pictorially extreme point solutions are the corner points of the set of feasible solutions. The following basic result shows that there is always an optimal extreme point solution to bounded linear programs.

**Lemma 1.2.2** *Let $P = \{x : Ax = b, x \geq 0\}$ and assume that the optimum value $\min\{c^T x : x \in P\}$ is finite. Then for any feasible solution $x \in P$, there exists an extreme point solution $x' \in P$ with $c^T x' \leq c^T x$.*

The following Rank Lemma is an important ingredient in the correctness proofs of almost all iterative algorithms in this monograph (see Chapter 2).

**Lemma 1.2.3 (Rank Lemma)** *Let $P = \{x : Ax = b, x \geq 0\}$ and let $x$ be an extreme point solution of $P$ such that $x_i > 0$ for each $i$. Then the number of variables is equal to the number of linearly independent constraints of $A$, i.e. the rank of $A$.*

### *1.2.1 Contradiction Proof Idea: Lower Bound > Upper Bound*

We give an outline of the proof that at each iteration there exists a variable with value 0 or 1. Suppose for contradiction that $0 < x_e < 1$ for every edge $e$. We use this assumption to derive a lower bound on the number of variables of the linear program. Let $n$ be the remaining vertices in $V_1$ (or $V_2$, they have the same cardinality) at the current iteration. Then each vertex in $V_1$ must have two edges incident on it, since $\sum_{v \in V_2:(u,v) \in E} x_{uv} = 1$ and $x_{uv} < 1$ for each $(u, v) \in E$. Thus the total number of edges is at least $2n$. This is a lower bound on the number of variables of the linear program, since we have one variable for each edge.

On the other hand, using the Rank Lemma, we derive an upper bound on the number of variables of the linear program. In the linear program for bipartite matching, we have only $2n$ constraints (one for each vertex in $V_1 \cup V_2$). Moreover, these $2n$ constraints are dependent since the sum of the constraints for vertices in $V_1$ equals the sum of the constraints for vertices in $V_2$. Hence, the number of linearly independent constraints is at most $2n-1$. By the Rank Lemma, the number of variables is at most $2n-1$. This provides us an upper bound on the number of variables. Since our upper bound is strictly smaller than the lower bound, we obtain the desired contradiction. Therefore, in an extreme point solution of the linear program for bipartite matching, there must exist a variable with value 0 or 1, and thus the iterative algorithm works. The number of iterations can be simply bounded by the number of edges in the bipartite graph. We give a formal proof of the above outline in Chapter 3.

### *1.2.2 Approximation Algorithms for NP-hard Problems*

The above framework can be naturally adapted to provide an approximation algorithm via the iterative method. In particular, for this, the iterative algorithm above typically has one or both of two additional steps: *Rounding* and *Relaxation*.

(i) **Rounding:** Fix a threshold $\alpha \geq 1$. If there is a variable $x_i$ that has a value of at least $\frac{1}{\alpha}$ in the optimal extreme point solution then pick the corresponding element in the solution being constructed.

(ii) **Relaxation:** Fix a threshold $\beta$. 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 in the residual formulation.

For the bipartite matching problem, we will see how the iterative algorithm presented above can be adapted to give approximation algorithms for the generalized assignment problem in Chapter 3. Other generalizations include the budgeted allocation problem in Chapter 3 and the hypergraph matching problem in Chapter 9.

## 1.3 Approach Outline

We now give an overview of the structure of the rest of the monograph. Early chapters in the book contain two main components: the first deals with proving the integrality of the LP relaxation of a well-studied problem, while the second shows how the iterative proof of integrality can be extended to design approximation algorithms for NP-hard variants of these basic problems. Both components follow the natural outline described below.

(i) **Linear Programming Formulation:** We start by giving a linear programming relaxation for the optimization problem we study. If the problem is polynomially solvable, this relaxation will be one with integral extreme points and that is what we will set out to show. If the problem is NP-hard, we state an approximation algorithmic result which we then set out to prove.

  (a) **Solvability:** Sometimes the linear programming relaxation we start with will be exponential in size. We then show that the linear program is solvable in polynomial time. Usually, this would entail providing a polynomial time *separation oracle* for the program using the formalism of the ellipsoid method [68]. Informally, the separation oracle is a procedure that certifies that any given candidate solution for the program is either feasible or not, and in the latter case provides a separating hyperplane which is a violated inequality of the formulation. In programs with an exponential number of such inequalities that are implicity described, the design of the separation oracle is itself a combinatorial optimization problem, and we sketch the reduction to one.

(ii) **Characterization of Extreme Point Solution:** We then give a characterization result for the optimal extreme point solutions of the linear program based on the Rank Lemma (Lemma 1.2.3). This part aims to show that any maximal set of linearly independent tight constraints at this extreme point solution can be captured by a sparse structure. Sometimes the proof of this requires the use of the *uncrossing* technique [30] in combinatorial optimization, which will be introduced in Chapter 4.

(iii) **Iterative Algorithm:** We present an iterative algorithm for constructing an integral solution to the problem from an extreme point solution. The algorithm has two simple steps.

   (a) If there is a variable in the optimal extreme point solution that is set to a value of 1, then include the element in the integral solution.
   (b) If there is a variable in the optimal extreme point solution that is set to a value of 0, then remove the corresponding element.

   In each of the above cases, at each iteration, we reduce the problem and arrive at a *residual* version, then we recompute an optimal extreme point solution and iterate the above steps until all variables have been set this way. In designing approximation algorithms we also use the rounding and relaxation steps as stated earlier.

(iv) **Analysis:** We then analyze the algorithm. This involves arguing the following two facts. First, we establish that the algorithm runs correctly and second, that it returns an optimal solution.

   (a) **Correctness:** We show that the iterative algorithm is correct by arguing that there is always an 1-element or a 0-element to pick in every iteration. This crucially uses the characterization of tight constraints at this optimal extreme point solution. The argument here also follows the same contradiction proof idea (lower bound $>$ upper bound): We assume for a contradiction that there is no 1-element or 0-element and get a large lower bound on the number of nonzero variables in the optimal extreme point solution. On the other side, we use the sparsity of the linearly independent tight constraints to show an upper bound on the number of such constraints. This then contradicts the Rank Lemma that insists that both these numbers are equal, and proves that there is always an 1- or 0-element.
   (b) **Optimality:** We finally show that the iterative algorithm indeed returns an optimal solution using a simple inductive argument. The crux of this argument is to show that the extreme point solution induced on the residual problem remains a feasible solution to this residual problem.

For the NP-hard variants of the problems we study, our goal is to show that the above framework can be naturally adapted to provide an approximation algorithm via the iterative method. In particular, recall that the iterative algorithm above typically has one or both of two additional steps: *Rounding* and *Relaxation*.

(i) **Rounding:** Fix a threshold $\alpha \geq 1$. If there is a variable $x_i$ which in the optimal extreme point solution has a value of at least $\frac{1}{\alpha}$ then include the corresponding element in the solution.

Adding this rounding step does not allow us to obtain optimal integral solution but only near-optimal solutions. Using the above step, typically one obtains an approximation ratio of $\alpha$ for covering problems addressed using this framework.

(ii) **Relaxation:** Fix a threshold $\beta$. 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 in the residual formulation.

The iterative relaxation step removes a constraint and hence this constraint can be violated in later iterations. But the condition on the removal of the constraints ensures that the constraint is only violated by an additive amount of $\beta$. This step enables us to obtain *additive* approximation algorithms for a variety of problems.

To summarize, for designing approximation algorithms, we first study the exact optimization problem in the above framework. We then use the above two steps in various combinations to derive strong approximation algorithms for constrained versions of these exact problems. In the last few chapters we find a few examples of approximation algorithms that do not strictly fit this framework (e.g. multicriteria versions, cut problems, bin packing) but the overall approach for these problems remains the same.

### 1.4 Book Chapters Overview

In the next chapter, we develop all the preliminaries needed in the following chapters. We discuss linear programs, and their polynomial time solvability using the separation oracle. We also outline the important Rank Lemma and other properties about the extreme point solutions. We also discuss the LP duality theorem and the complementary slackness conditions, and some basic facts about submodular functions and graphs.

A first stream of chapters study problems in undirected graphs. In Chapter 3, we give the first example to illustrate the iterative method on bipartite matching and vertex cover problems. We also show how the proof for the bipartite matching leads to approximation algorithms for the generalized assignment problem and the budgeted allocation problem. In Chapter 4, we study the classical spanning tree problem and its extension to the minimum bounded degree spanning tree problem. This chapter introduces the uncrossing technique in combinatorial optimization. In Chapter 5, we generalize the arguments for undirected spanning trees to bases of matroids as well as to the common bases in the intersection of two matroids, and also to the minimum bounded degree matroid basis problem and the maximum common independent set problem in the intersection of $k$ matroids. We also show integrality of the dual of matroid and matroid intersection problems which lead to certain min-max results.

A second stream of chapters study problems in directed graphs. In Chapter 6, we

study the directed rooted spanning tree (or arborescence) problem, along with a degree-bounded version, and then generalize the method developed here to a rooted $k$-connected subgraph problem providing a self-contained proof of a result of Frank and Tardos. This is developed further in Chapter 7 to showing the integrality of submodular flow problems. For this last problem, we again complement the proof of exact LP characterization with a description of an approximation algorithm for the degree-bounded version built upon the proof of the exact counterpart. For the submodular flow problem, we also give a proof of the integrality of its dual.

We then present a few more advanced chapters applying the iterative method. In Chapter 8, we apply the iterative method to general problems involving network matrices as constraint matrices (with integral right hand sides) and their duals. We then show the application of network matrices to derive integrality of the duals of various linear programs encountered in earlier chapters (such as those for matroid bases, matroid intersection and submodular flow). In Chapter 9, we address the generalization of perfect and maximum matchings in bipartite graphs to general graphs, and also address higher dimensional matching problems and other matching variants. We then present a common generalization of Jain's 2-approximation algorithm for the survivable network design problem (SNDP), and a result of Boyd and Pulleyblank on 1-edges in the Held-Karp relaxation for the Symmetric Traveling Salesman Problem (STSP) in Chapter 10. This chapter also generalizes Jain's result to degree bounded network design problems. In Chapter 11, we extend the application of the method to constrained optimization problems such as partial covering and multicriteria problems. In Chapter 12, we add the primal-dual complementary slackness conditions to the iterative method to derive approximation results for some cut problems. In Chapter 13 we present some early examples of iterative methods, including the Beck-Fiala theorem on discrepancy and Karmarkar-Karp algorithm for bin packing. Most chapters contain selected historical notes as well as exercises.

## 1.5 Historical Remarks

Polyhedral combinatorics, the compact polyhedral description of important combinatorial optimization problems, is a fundamental and unifying tool in algorithms, combinatorics and optimization. A highlight of this line of research is the pioneer work by Jack Edmonds [37]; we refer the reader to the book [130] and the historical survey [128] by Schrijver for an encyclopedic treatment of this subject.

In approximation algorithms, the first use of the iterative rounding method is due to Jain [78], and the first uses of the iterative relaxation method can be traced back to the proof of a discrepancy theorem by Beck and Fiala [18] and the approximation algorithm for the bin packing problem by Karmarkar and Karp [80].

# 2

# Preliminaries

In this chapter we discuss linear programming and basic facts about extreme point solutions to linear programs. We then briefly discuss solution methods for linear programs, particularly stating the sufficiency of finding a separation oracle for the program to be able to solve it. We then state some concepts from graph theory which are used throughout the book. The last part of the chapter discusses submodular and supermodular functions. These functions give a general tool for modeling a variety of optimization problems. Excellent introductory textbooks or surveys in all three areas are available for further reference [16, 76, 139].

## 2.1 Linear Programming

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 a *feasible* solution. If there exists a feasible solution to the linear program, 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 linear program is *unbounded* (from below) if $\forall \lambda \in \mathbb{R}, \exists$ feasible $x$ such that $c^T x < \lambda$.

There are different forms in which a linear program can be represented. However, all these forms are equivalent to the form we consider above and can be converted into one another by simple linear transformations (see e.g., [29]).

### *2.1.1 Extreme Point Solutions to Linear Programs*

In this subsection, we discuss basic properties about extreme point solutions. First, we have the following definition.

**Definition 2.1.1** *Let $P$ be a polytope and let $x$ be an extreme point solution of $P$ then $x$ is **integral** if each co-ordinate of $x$ is an integer. The polytope $P$ is called **integral** if every extreme point of $P$ is integral.*

We now show basic properties about extreme point (or vertex) solutions. Most proofs are quite standard and we give a short sketch. The reader is referred to standard texts on Linear Programming (e.g. Chvatal [29]) for details. We now prove Lemma 1.2.2. We state it again for completeness.

**Lemma 2.1.2** *Let $P = \{x : Ax \geq b, x \geq 0\}$ and assume that $\min\{c^T x : x \in P\}$ is finite. Then for every $x \in P$, there exists an extreme point solution $x' \in P$ such that $c^T x' \leq c^T x$, i.e., there is always an extreme point optimal solution.*

*Proof* The idea of the proof is to show by how we can move from a current optimal solution to one that has more zero components or more tight constraints and is thus closer to being an extreme point solution.

Consider $x$ such that it is optimal but not an extreme point solution. 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 rows which are at equality at $x$, and $b^=$ be the vector $b$ restricted to these rows. Thus 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 use $-y$). Consider $x + \lambda y$ for $\lambda > 0$ and increase $\lambda$ until $x + \lambda y$ is no longer feasible due to the non-negativity constraints. Formally, let

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

We now show that $x+\lambda^* y$ 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. Since we assume that $\min\{c^T x : x \in P\}$ is finite, $\lambda^*$

is finite and the solution $x + \lambda^* y$ has one more zero coordinate (when $\lambda^* = (x_j)/(-y_j)$) or one extra tight constraint (when $\lambda^* = (A_i x - b_i)/(-A_i y)$).

Proceeding this way, we can convert any optimal solution to one that is also an extreme point solution, proving the claim. □

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

**Lemma 2.1.3** *Let $P = \{x : Ax \geq b, x \geq 0\}$. For $x \in P$, let $A^=$ be the submatrix of $A$ restricted to rows which are at equality at $x$, and let $A^=_x$ denote the submatrix of $A^=$ consisting of the columns corresponding to the nonzeros in $x$. Then $x$ is an extreme point solution if and only if $A^=_x$ has linearly independent columns (i.e., $A^=_x$ has full column rank).*

*Proof* ($\Leftarrow$) If $x$ is not an extreme point solution, 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 theorem). 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 an extreme point solution. 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 $x_j = 0$. Note that there exists $\epsilon$ such that $x + \epsilon y \geq 0$ and $x - \epsilon y \geq 0$. Also $x + y$ and $x - y$ 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 an extreme point solution. □

We now prove the important Rank Lemma. We restate the Lemma (in canonical form) for completeness.

**Lemma 2.1.4 (Rank Lemma)** *Let $P = \{x : Ax \geq b, x \geq 0\}$ and let $x$ be an extreme point solution of $P$ such that $x_i > 0$ for each $i$. Then any maximal number of linearly independent tight constraints of the 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 2.1.3 it follows that $A^=$ has full column rank. Since the number of columns equals the number of non-zero variables in $x$ and row rank of any matrix equals the column rank†, we have that 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. □

† Important check: If you are rusty on why this statement is true, a crisp proof of the equality of the row rank and column rank can be found in the short note due to Andrea and Wong [3] that is available on the web.

Next, we highlight various methods of solving linear programs. First we introduce the concept of *basic feasible solutions* and show their equivalence to extreme point solutions. Basic feasible solutions form a key ingredient in the Simplex algorithm which is the most widely used algorithm for solving linear programs in practice.

#### *2.1.1.1 Basic Feasible Solution*

Consider the linear program

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

By introducing slack variables $s_j$ for each constraint, we obtain an equivalent linear program in *standard form.*

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

Henceforth, we study linear program in standard form: $\{\min cx : Ax = b, x \geq 0\}$. Without loss of generality, we can assume that $A$ is of full row rank. If there are dependent constraints, we can remove them without affecting the system or its optimal solution.

A subset of columns $B$ of the constraint matrix $A$ is called a *basis* if the matrix of columns corresponding to $B$, i.e. $A_B$, is invertible. A solution $x$ is called *basic* if and only if there is a basis $B$ such that $x_j = 0$ if $j \notin B$ and $x_B = A_B^{-1}b$. If in addition to being basic, it is also feasible, i.e., $A_B^{-1}b \geq 0$, it is called a *basic feasible solution* for short. The correspondence between bases and basic feasible solutions is not one to one. Indeed there can be many bases which correspond to the same basic feasible solution. The next theorem shows the equivalence of extreme point solutions and basic feasible solutions.

**Theorem 2.1.5** *Let $A$ be a $m \times n$ matrix with full row rank. Then every feasible $x$ to $P = \{x : Ax = b, x \geq 0\}$ is a basic feasible solution if and only if $x$ is an extreme point solution.*

*Proof* ($\Rightarrow$) If $x$ is a basic feasible solution, then $A_B$ is invertible. Since $A_x$ is a submatrix of $A_B$ (it is a proper submatrix if some basic variable is at 0), $A_x$ has linearly independent columns. Therefore, by Lemma 2.1.3, $x$ is an extreme point solution.

($\Leftarrow$) If $x$ is an extreme point solution, then by Lemma 2.1.3, $A_x$ has linearly independent columns. Now we can add columns to $A_x$ from $A$ to convert it into an invertible matrix $A_B$. Note that since $Ax = b$, $A_B x_B + A_N x_N = b$, where $A_N$ and $x_N$ denote the

non-basic parts of $A$ and $x$ respectively. By construction of $A_B$, $x_N = 0$ and so $x_B = A_B^{-1}b$. So $x$ is a basic feasible solution with $A_B$ as the basis. □

### *2.1.2 Algorithms for Linear Programming*

The *simplex* algorithm solves linear programs to get a basic feasible optimal solution. It works by starting at any basic feasible solution and moving to a *neighboring* basic feasible solution which improves the objective function. The *convexity* of the linear program ensures that once the simplex algorithm ends at a *local* optimum basic feasible point, it has achieved the global optimum as well. Many variants of the simplex algorithm have been considered, each defined by which neighboring basic feasible solution to move in case there are more than one improving basic feasible points in the neighborhood. Although the simplex algorithm works efficiently in practice, there are examples where each variant of the simplex algorithm runs in exponential time. Again, for more details, see e.g. [29].

Polynomial-time algorithms for solving linear programs fall in two categories: ellipsoid algorithms [82] and interior point algorithms [79]. We refer the reader to Nemhauser and Wolsey [113] and Wright [143] for details about these algorithms. Both these algorithms solve linear programs to obtain *near* optimal solution in polynomial time. Moreover, there are rounding algorithms [113] which, given a sufficiently *near* optimal solution to a linear program, return an optimal extreme point solution.

**Theorem 2.1.6** *There is an algorithm which returns an optimal extreme point solution to a linear program. Moreover, the running time of the algorithm is polynomial in the size of the linear program.*

### *2.1.3 Separation and Optimization*

In this book, we will also 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 enumerate them, let alone solve them in polynomial time. We use the notion of separation to show that many exponentially sized linear programs can be solved in polynomial time.

**Definition 2.1.7** *Given $x^* \in \mathcal{R}^n$ and a polytope $P = \{x : Ax \geq b, x \geq 0\}$, 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 $A_i x \geq b_i$ for $P$ which is violated by $x^*$, i.e., $A_i x^* < b_i$.*

The following theorem of Grotschel, Lóvasz and Schrijver [68] shows that polynomial time separability is equivalent to polynomial time solvability of a linear program; we state

it in a form that is convenient for combinatorial optimization problems. The basis of this equivalence is the ellipsoid algorithm.

**Theorem 2.1.8** *Given a full-dimensional polytope $P$ and a polynomial-time separation oracle for $P$, one can find an optimal extreme point solution to a linear objective function over $P$ (assuming it is bounded) via the Ellipsoid algorithm that uses a polynomial number of operations and calls to the separation oracle.*

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 too slow. In this book, as we consider LP formulations with an exponential number of constraints, we will often provide efficient separation oracles showing that the linear program for the problem is solvable in polynomial time.

### *2.1.4 Linear Programming Duality*

Linear programming duality is a key concept to certify and characterize optimal solutions to linear programs. Consider the following primal linear program in the standard form:

$$\begin{aligned}
\text{minimize} \quad & \sum_{j=1}^{n} c_j x_j & & & \\
\text{subject to} \quad & \sum_{j=1}^{n} a_{ij} x_j & \geq & b_i & \forall\, 1 \leq i \leq m \\
& x_j & \geq & 0 & \forall\, 1 \leq j \leq n
\end{aligned}$$

The corresponding dual program is

$$\begin{aligned}
\text{maximize} \quad & \sum_{i=1}^{m} b_i y_i & & & \\
\text{subject to} \quad & \sum_{i=1}^{m} a_{ij} y_i & \leq & c_j & \forall\, 1 \leq j \leq n \\
& y_i & \geq & 0 & \forall\, 1 \leq i \leq m
\end{aligned}$$

It is not difficult to show that the optimal value of the primal linear program is at least the optimal value of the dual linear program, and thus any dual feasible solution provides a lower bound on the optimal value of the primal program. This is called the weak LP duality theorem, whose proof also follows from the derivation of the complementary slackness conditions below. A fundamental result in linear programming is the strong duality theorem, which shows that the optimal value of the primal linear program is actually equal to that of the dual linear program.

**Theorem 2.1.9 (Strong Duality Theorem)** *If the primal linear program has an optimal solution, so does its dual, and the respective optimal costs are equal.*

Many combinatorial min-max theorems can be derived from the strong duality theorem. For example, we will see in Chapter 3 the min-max theorem for bipartite matching, and in Chapter 5 for the min-max theorem for matroid intersection. We refer the reader to any textbook on linear programming (e.g. [29]) for the proof of the strong duality theorem.

#### *2.1.4.1 Complementary Slackness Conditions*

The complementary slackness conditions provide a characterization for an optimal primal solution $x$ and an optimal dual solution $y$. We will use the complementary slackness conditions in Chapter 12.

**Primal complementary slackness conditions:**

$$\text{Either } x_j = 0 \text{ or } \sum_{i=1}^{m} a_{ij} y_i = c_j.$$

**Dual complementary slackness conditions:**

$$\text{Either } y_i = 0 \text{ or } \sum_{j=1}^{n} a_{ij} x_j = b_i.$$

These conditions can be derived as follows:

$$\begin{aligned} \sum_{j=1}^{n} c_j x_j &\geq \sum_{j=1}^{n} \big(\sum_{i=1}^{m} a_{ij} y_i\big) x_j \\ &= \sum_{i=1}^{m} \big(\sum_{j=1}^{n} a_{ij} x_j\big) y_i \\ &\geq \sum_{i=1}^{m} b_i y_i, \end{aligned}$$

where the first inequality is by the constraints in the dual linear program, the second equality is by interchanging the order of the summations, and the third inequality is by the constraints in the primal linear program. Note that this shows the weak duality theorem. Since $x$ and $y$ are optimal solutions, by the strong duality theorem, we have that $\sum_{j=1}^{n} c_j x_j = \sum_{i=1}^{m} b_i y_i$, and thus equality must hold throughout. The primal complementary slackness conditions follow from the first inequality holding as an equality, while the dual complementary slackness conditions follow from the last inequality holding as an equality.

## 2.2 Graphs and Digraphs

Most problems addressed in this book are on networks connecting nodes with edges or links. We define graph theoretic concepts which will be encountered in later chapters.

Given an undirected graph $G = (V, E)$ and a set $S \subseteq V$, we denote $\delta_G(S)$ or $\delta_E(S)$ be the set of edges which have exactly one endpoint in $S$. For a vertex $v \in V$, $\delta_G(\{v\})$ is simply denoted by $\delta_G(v)$. We also denote $d_G(v)$ or $d_E(v)$ to be degree of $v$, i.e., $|\delta_G(v)|$. For sets $X, Y \subseteq V$, we denote $E_G(X, Y)$ to be the set of edges which has exactly one endpoint in $X$ and one in $Y$. We also denote $E_G(X, X)$ by $E_G(X)$. Observe that $\delta_G(X) = E_G(X, V \setminus X)$. We also denote $|\delta_G(X)|$ by $d_G(X)$. The subscript $G$ or $E$ is sometimes dropped from the notation if the graph $G$ is clear from the context. A subgraph $H$ of $G$ is *spanning* if it contains all the vertices of $G$, *connected* if there is a path between any two vertices of $H$, a *tree* if it is acyclic and connected. An important concept is a *spanning tree*, subgraph which is both spanning and a tree. Observe that a spanning tree is a also a minimally spanning connected subgraph.

Given a directed graph $D = (V, A)$ and a set $S \subset V$, we denote $\delta^{in}_D(S)$ to be the set of arcs whose head is in $S$ but tail is not in $S$. Similarly, $\delta^{out}_D(S)$ is the set of arcs whose tail is in $S$ but the head is not in $S$. For a vertex $v \in V$, we denote $\delta^{in}_D(\{v\})$ as $\delta^{in}_D(v)$ and $\delta^{out}_D(\{v\})$ as $\delta^{out}_D(v)$. The in-degree of $v$, $|\delta^{in}_D(v)|$ is denoted by $d^{in}_D(v)$ and the out-degree of $v$, $|\delta^{out}_D(v)|$ is denoted by $d^{out}_D(v)$. The degree of $v$, $d_D(v)$ is the sum of its in-degree and out-degree. For sets $X, Y \subseteq V$, we denote $E_D(X, Y)$ to be the set of arcs whose tail is in $X$ and the head is in $Y$. Observe that $\delta^{out}_D(X) = E_D(X, V \setminus X)$. We denote $|\delta^{in}_D(X)|$ and $|\delta^{out}_D(X)|$ by $d^{in}_D(X)$ and $d^{out}_D(X)$ respectively. A subgraph $H$ of $D$ is called *strongly connected* if there is a directed path from each vertex of $H$ to every other vertex, *weakly connected* if the underlying undirected graph is connected, acyclic if there is no directed cycle in $H$. $H$ is called an *arborescence* the underlying graph is a spanning tree and the graph has only one vertex with no in-edges which is called the root. If the root of an arborescence is the vertex $r$ then it is also called an $r$-arborescence.

We will use frequently the following fundamental min-max theorem in graph theory.

**Theorem 2.2.1 (Menger's Theorem [105])** *Let $D = (V, A)$ be a directed graph, and $s, t \in V$ be two distinct vertices. The maximum number of arc-disjoint $s$-$t$ paths in $D$ is equal to the minimum $d^{in}_D(X)$ over all $X \subset V$ with $s \notin X$ and $t \in X$.*

Menger's theorem shows a close connection between disjoint paths and cuts, which will be used in writing linear programs and constructing separation oracles. One can also obtain the corresponding min-max theorems for undirected graphs and for vertex connectivity by simple transformations (see the exercises).

## 2.3 Submodular and Supermodular Functions

In this section, we define special classes of set functions with some nice convexity-like properties. Typically, in our applications, these functions are defined over a set of vertices of a graph we will be working with; most of the time, they will also be integer valued and

positive. More comprehensive treatments on these topics are available in the monograph by Fujishige [53] and the book by Schrijver [130].

### *2.3.1 Submodularity*

**Definition 2.3.1** *A function $f : 2^V \to \mathbb{R}$ is submodular if for every pair $A, B$ of subsets of $V$, we have*

$$f(A) + f(B) \geq f(A \cap B) + f(A \cup B).$$

A simple example of a submodular set function defined on the vertices of an undirected graph $G = (V, E)$ is cardinality of the cut function $d : 2^V \to \mathbb{Z}_+$ where $d(S) = |\delta(S)|$.

**Proposition 2.3.2** *The cut function $d$ of any undirected graph is submodular.*

*Proof* To see that $d$ is submodular, note that on the right hand side, we have

$$d(A \cap B) = |E(A \cap B, A \setminus B)| + |E(A \cap B, B \setminus A)| + |E(A \cap B, V \setminus (A \cup B))|.$$

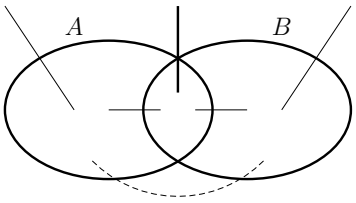


Fig. 2.1. In this example the solid edges are counted exactly once in both the LHS $d(A) + d(B)$, and the RHS $d(A \cap B) + d(A \cup B)$, and the bold edge is counted exactly twice on both sides. The dashed edge is counted in LHS but not in RHS.

Similarly we also have

$$d(A \cup B) = |E(A \cap B, V \setminus (A \cup B))| + |E(A \setminus B, V \setminus (A \cup B))| + |E(B \setminus A, V \setminus (A \cup B))|.$$

On the left hand side we have

$$d(A) = |E(A \cap B, V \setminus (A \cup B))| + |E(A \cap B, B \setminus A)| + |E(A \setminus B, V \setminus (A \cup B))| + |E(A \setminus B, B \setminus A)|.$$

Similarly we get

$$d(B) = |E(A \cap B, V \setminus (A \cup B))| + |E(A \cap B, A \setminus B)| + |E(B \setminus A, V \setminus (A \cup B))| + |E(B \setminus A, A \setminus B)|.$$

Comparing the above expressions shows that the edges in $E(A \setminus B, B \setminus A)$ are responsible for the inequality (rather than equality). Also see Figure 2.1. □

Note that the edge cut function can be extended to the case when nonnegative weights, say $x : E \to \mathbb{R}_+$ are assigned to the edges. Instead of $d(S) = |\delta(S)|$, we have $x(\delta(S)) = \sum_{e \in \delta(S)} x_e$. The above proof also shows that the function $x$ is submodular as well.

**Proposition 2.3.3** *The weighted-cut function of any undirected graph is submodular.*

Using the same method as above, it is not hard to also verify that we also have

$$d(A) + d(B) \geq d(A \setminus B) + d(B \setminus A). \tag{2.1}$$

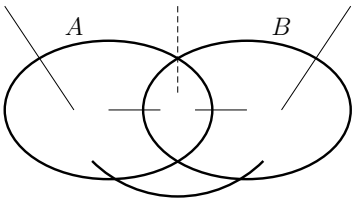


Fig. 2.2. In this example the solid edges are counted exactly once in both the LHS $d(A) + d(B)$, and the RHS $d(A \setminus B) + d(B \setminus A)$, and the bold edge is counted exactly twice on both sides. The dashed edge, however is counted in the LHS but not in the RHS.

Let us define a stronger notion of submodularity to capture these properties of edge cut function.

**Definition 2.3.4** *A function $f : 2^V \to \mathbb{R}$ is strongly submodular if for every pair $A, B$ of subsets of $V$, we have*

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

*and*

$$f(A) + f(B) \geq f(A \setminus B) + f(B \setminus A).$$

As noted above, edge cut functions in undirected graphs are strongly submodular. The second property in the definition above has also been referred to as *posimodularity* [108, 109].

### *2.3.2 Supermodularity*

We move on to a symmetric concept, supermodularity.

**Definition 2.3.5** *A function $f : 2^V \to \mathbb{R}$ is supermodular if for every pair $A, B$ of subsets of $V$, we have*

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

As before, a simple example of a supermodular set function defined on the vertices of an undirected graph $G = (V, E)$ is the induced edge function $d : 2^V \to \mathbb{Z}_+$ where $i(S)$ is the number of edges in $E$ with both endpoints in $S$, i.e., $i(S) = |E(S)|$. A verification similar to the above can be carried out to establish that the induced edge function is supermodular. Also, if nonnegative values, say $x : E \to \mathbb{R}_+$ are assigned to the edges and we consider $x(S) = \sum_{e \in E(S)} x_e$, it follows that this function is supermodular as well.

**Proposition 2.3.6** *The induced edge function $i(.)$ for any undirected graph is supermodular. This is also true for the weighted version with nonnegative weights.*

### *2.3.3 Refinements*

**Definition 2.3.7** *Two subsets $A$ and $B$ of a ground set $V$ are* intersecting *if $A \cap B \neq \emptyset$. A function $f : 2^V \to \mathbb{R}$ is* intersecting-submodular *if for every pair $A, B$ of intersecting subsets of $V$, we have*

$$f(A) + f(B) \geq f(A \cap B) + f(A \cup B).$$

**Definition 2.3.8** *Two subsets $A$ and $B$ of a ground set $V$ are* crossing *if none of the four subsets $A \cap B, A \setminus B, B \setminus A$ and $V \setminus (A \cup B)$ are empty. A function $f : 2^V \to \mathbb{R}$ is* crossing-submodular *if for every pair $A, B$ of crossing subsets of $V$, we have*

$$f(A) + f(B) \geq f(A \cap B) + f(A \cup B).$$

To distinguish the regular submodular functions from the more restricted intersecting and crossing varieties, they are also sometimes dubbed fully submodular. Other important examples of fully submodular functions arises as cut functions of directed graphs $D = (V, A)$. Define $\delta^{in}(S)$ for a subset $S$ of vertices as the set of arcs whose heads are in $S$ and tails are in $V \setminus S$, which we can denote as $A(V \setminus S, S)$. Symmetrically, define $\delta^{out}(S)$ as the set $A(S, V \setminus S)$. Denote $|\delta^{in}(S)|$ and $|\delta^{out}(S)|$ by $d^{in}(S)$ and $d^{out}(S)$ respectively. Both these functions $d^{in}$ and $d^{out}$ defined on vertex subsets are fully submodular; see Figure 2.3. Unlike undirected graphs, however, the functions $d^{in}$ and $d^{out}$ are not strongly submodular.

**Proposition 2.3.9** *The cut functions $d^{in}$ and $d^{out}$ of any directed graph are submodular. This is also true for the weighted directed cut functions with non-negative weights.*

A broader class of functions generalizing supermodularity is useful in specifying connectivity requirements for network design problems.

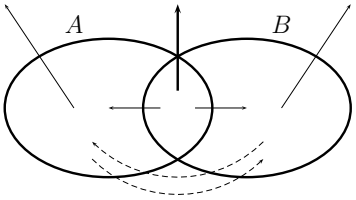


Fig. 2.3. In this example the solid arcs are counted exactly once in both the LHS $\delta^{out}(A)+\delta^{out}(B)$ and the RHS $\delta^{out}(A\cup B)+\delta^{out}(A\cap B)$ and the bold edges are counted exactly twice on both sides. The dashed edges are counted in the LHS but not in the RHS.

**Definition 2.3.10** *A function $f : 2^V \rightarrow \mathbb{R}$ is skew (or weakly) supermodular if for every pair $A, B$ of subsets of $V$, at least one of the following inequalities is true.*

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

$$f(A) + f(B) \leq f(A \setminus B) + f(B \setminus A)$$

In the survivable network design problem on an undirected graph $G = (V, E)$, we are given nonnegative integral edge-connectivity requirements $r_{uv}$ for all pairs of vertices, and we are interested in finding a subgraph with at least $r_{uv}$ edge-disjoint paths between $u$ and $v$ for every pair $u, v$. If we define the connectivity requirement function $f$ for any set $S$ as $f(S) = max_{u\in S, v\notin S}\ r_{uv}$, it is not hard to verify that $f$ is skew supermodular.

Functions that are both submodular and supermoduar are modular - the typical example being the cardinality (or "modulus") function. Furthermore, $f$ is submodular iff $-f$ is supermodular, and if $f$ is submodular and $g$ is supermodular, then $f - g$ is submodular. Finally, if $f$ is skew supermodular and $g$ is strongly submodular, then $f - g$ is weakly supermodular. These follow directly as a consequence of the definitions.

#### *2.3.3.1 Minimizing submodular function*

A rich literature examines the minimization of submodular functions in (strongly) polynomial time - Chapter 45 of the book by Schrijver [130] contains most references on this topic. We mention three important results in this vein: Queyranne [117] gave an algorithm for minimizing symmetric submodular functions, building on earlier work of Nagamochi and Ibaraki for finding a minimum cut in an undirected graph. Grotschel, Lovasz and Schrijver [69] gave a strongly polynomial time algorithm for finding the minimum of a submodular function only over sets of odd cardinality, building on earlier work of Padberg and Rao. Finally, for the general case of submodular function with only oracle access to values, strongly polynomial time algorithms were first presented by Iwata, Fleischer and Fujishige [77] and also independently by Schrijver [127]. More generally defined submodu-

lar function (e.g., on intersecting or cross-free families) can also be minimized in polynomial time - see Chapter 49 in [130].

We will be making use of submodular (and supermodular) functions extensively in this monograph. In a typical application, the set of tight constraints at an extreme point solution corresponds to a set of cuts in a graph for which a (submodular) function value is an integer. Depending on the type of submodular function (full, intersecting or crossing), these tight constraints can then be *uncrossed* to result in a nicely structured basis for them: For instances, for fully submodular constraints, the basis is a chain; for intersecting submodular constraint forms, the basis is a laminar family representable by the forest capturing set-inclusion relation; for crossing submodular constraint systems, the basis is a cross-free family which can also be turned into a laminar family. Such structured sparse representations of the tight constraints at any extreme point solution are the key to proving integrality for many problems – they pave the way to show the upper bound part of the general argument outlined in the beginning of this chapter, typically by a counting argument that is carried out inductively on a representation of these sparse families.

**Exercises**

2.1 Consider a bounded linear program. Prove that an LP solution $x$ is a basic feasible solution (or extreme point solution) if and only if there is an objective function $c$ such that $x$ is the unique optimum solution.

2.2 Prove Menger's theorem for undirected graphs: The maximum number of edge-disjoint $s$-$t$ paths is equal to the minimum $d(X)$ over all sets $X \subset V$ with $s \notin X$ and $t \in X$. (Hint: "bidirect" the edges and apply Theorem 2.2.1.)

2.3 Two directed $s$-$t$ paths $P_1$ and $P_2$ are internally vertex-disjoint if $V(P_1) \cap V(P_2) = \{s, t\}$. A vertex set $U \subset V$ is an $s$-$t$ cut if there is no directed $s$-$t$ paths in $G - U$. Prove Menger's theorem for vertex connectivity: The maximum number of internally vertex-disjoint $s$-$t$ paths is equal to the minimum size of a vertex $s$-$t$ cut. (Hint: "split" each vertex appropriately and apply Theorem 2.2.1.)

2.4 Derive a corresponding Menger's theorem for vertex connectivity in undirected graphs.

2.5 Verify Proposition 2.3.9 that the in- and out-cut functions $d^{in}$ and $d^{out}$ for a digraph are indeed submodular but not strongly submodular.

2.6 Verify Proposition 2.3.6. Is the induced edges (arcs) function supermodular for digraphs?

2.7 Show that a function $f : 2^V \rightarrow R_+$ is (fully) submodular if and only if $f(X + v) - f(X) \geq f(Y + v) - f(Y)$ whenever $X \subseteq Y$ and $v \in V - X$.

2.8 Use the above equivalent definition of submodular function to derive Proposition 2.3.2 and Proposition 2.3.9.

2.9 Show that the connectivity requirement function for the survivable network design problem, $f(S) = max_{u \in S, v \notin S}\ r_{uv}$ for all $S \subset V$ is skew supermodular.

2.10 Show that the connectivity requirement function for the $k$-connected subgraph problem, $f(S) = k$ for every non-empty $S \subset V$ is crossing supermodular.

2.11 Show that the connectivity requirement function for the rooted-connectivity problem, $f(S) = k$ for every non-empty $S \subset V$ with $r \notin S$ for a specified vertex $r$ is intersecting supermodular.

# 3

# Matching and Vertex Cover in Bipartite Graphs

In this chapter we consider two very closely related problems, maximum weighted matching and minimum cost vertex cover in bipartite graphs. Linear programming duality plays a crucial role in understanding the relationship between these problems. We will show that the natural linear programming relaxations for both the matching problem and the vertex cover problem are integral, and then use duality to obtain a min-max relation between them. Nevertheless, our proofs of integrality use the iterative method by arguing the existence of 1-elements in an extreme point solution.

In the first section, we show the integrality of the more standard maximization version of the matching problem introduced in the previous chapter. In the following sections, we show two applications of the proof technique for integrality to derive approximation results for NP-hard problems. We first present a new proof of an approximation result for the generalized assignment problem, and then present an approximation result for the budgeted allocation problem. The proofs of both of these results develop on the integrality result for the bipartite matching problem. Following this, we discuss the integrality of the bipartite vertex cover problem formulation and conclude with a short section on the duality relation between these problems and some historical notes.

## 3.1 Matchings in Bipartite Graphs

In this section, we show that the matching polytope in bipartite graphs is integral. Given a bipartite graph $G = (V_1 \cup V_2, E)$ and a weight function $w : E \to \mathcal{R}$, the maximum matching problem is to find a set of vertex-disjoint edges of maximum total weight.

### *3.1.1 Linear Programming Relaxation*

The linear programming relaxation for the bipartite matching problem is given by the following $LP_{bm}$. In the following $\delta(v)$ denotes the set of edges incident to $v$.

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

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 Theorem 2.1.6.

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

**Theorem 3.1.1** *Given any weight function $w$ there exists an integral matching $M$ such that $w(M) \geq w \cdot x$ where $x$ is an optimal solution to $LP_{bm}(G)$.*

Our proof of Theorem 3.1.1 as a corollary implies the following theorem.

**Theorem 3.1.2** *The linear programming formulation $LP_{bm}(G)$ is integral.*

### *3.1.2 Characterization of Extreme Point Solutions*

Before we prove Theorem 3.1.1 we give a characterization of extreme point solutions of $LP_{bm}(G)$ for which we need a few definitions.

For a set $F \subseteq E$, let $\chi(F)$ denote the vector in $\mathbb{R}^{|E|}$ that has an 1 corresponding to each edge $e \in F$, and 0 otherwise. This vector is called the *characteristic vector* of $F$. In the following lemma which follows by a direct application of the Rank Lemma 1.2.3, we characterize an extreme point solution by a set of tight linearly independent constraints.

**Lemma 3.1.3** *Given any extreme point solution $x$ to $LP_{bm}(G)$ such that $x_e > 0$ for each $e \in E$ there exists $W \subseteq V_1 \cup V_2$ such that*

(i) $x(\delta(v)) = 1$ *for each* $v \in W$.
(ii) *The vectors* $\{\chi(\delta(v)) : v \in W\}$ *are linearly independent.*
(iii) $|W| = |E|$.

### *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 3.1.1. The algorithm is a simple iterative procedure as shown in Figure 3.1.

**Iterative Bipartite Matching Algorithm**

(i) Initialization $F \leftarrow \emptyset$.
(ii) While $E(G) \neq \emptyset$ do
  (a) Find an optimal extreme point solution $x$ to $LP_{bm}(G)$ 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 $F \leftarrow F \cup \{e\}$ and $G \leftarrow G \setminus \{u, v\}$.
(iii) Return $F$.

Fig. 3.1. Bipartite Matching Algorithm

### *3.1.4 Correctness and Optimality*

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 (ii)(a) or an edge $e$ with $x_e = 1$ in Step (ii)(b). In the second part, we show that the algorithm will always find such an edge completing the proof.

**Claim 3.1.4** *If the algorithm, in every iteration, finds an edge $e$ with $x_e = 0$ in Step* (ii)(a) *or an edge $e$ with $x_e = 1$ in Step* (ii)(b)*, 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 (ii)(a) of the algorithm, then the residual problem is to find a matching in the graph $G \setminus \{e\}$, where we remove the edge $e$ from $G$. The residual solution $x_{res}$, $x$ restricted to $G \setminus \{e\}$, 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 (ii)(b) 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 $G' = G \setminus \{u, v\}$, where we remove the vertices $u$ and $v$ and their incident edges from $G$. 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 $w(F') \geq w \cdot x_{res}$, as $x_{res}$ 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 3.1.1 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 extreme point solutions given in Lemma 3.1.3.

**Lemma 3.1.5** *Given any extreme point 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 3.1.3 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 3.1.6** *We must have $d_E(v) = 2$ for each $v \in W$ and $d_E(v) = 0$ for each $v \notin W$.*

*Proof* Firstly, $d_E(v) \geq 2$ for each $v \in W$, since $x(\delta(v)) = 1$ for each $v$ and $0 < x_e < 1$ for each $e \in E$. This implies that:

$$2|W| = 2|E| = \sum_{v \in V} d_E(v) \geq \sum_{v \in W} d_E(v) \geq 2|W|.$$

This implies the inequalities must hold as equalities, and thus $d_E(v) = 0$ for each $v \notin W$ by the first inequality and $d_E(v) = 2$ for each $v \in W$ by the second inequality. □

Hence, $E$ is a cycle cover on the 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 (ii) of Lemma 3.1.3. Therefore any extreme point solution $x$ to $LP_{bm}(G)$ must have an edge $e$ with $x_e = 0$ or $x_e = 1$. □

Thus we obtain from Lemma 3.1.5 that the algorithm in Figure 3.1 returns a matching with total weight at least the weight of the linear program. This completes the proof of Theorem 3.1.1.

## 3.2 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

following scheduling problem on unrelated parallel machines with costs. 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 $c_{ij}$. 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.

Shmoys and Tardos [131] gave an algorithm which returns an assignment of cost at most $C$ and each machine is used for 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 [131] using the iterative relaxation method. This proof develops on the iterative proof of the integrality of the bipartite matching given in Section 3.1. We shall prove the following theorem.

**Theorem 3.2.1** *There exists a polynomial time algorithm for the generalized assignment problem which returns a solution of cost at most $C$ that uses each machine $i$ for at most $2T_i$ time units, where $C$ is the cost of an optimal assignment that uses each machine $i$ for at most $T_i$ time units.*

### *3.2.1 Linear Programming Relaxation*

Before we write the linear program for the problem, we first model the problem as a bipartite matching problem. We start with a complete bipartite graph $G$ 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 $d_F(j) = 1$ for each job $j \in J$, so that the edge incident at $j$ denotes which machine job $j$ is assigned to. The time constraint at machines can be modeled by restricting that $\sum_{e \in \delta(i) \cap F} p_{ij} \leq T_i$ for each machine $i$. We strengthen this model by disallowing certain assignments using the following observation: If $p_{ij} > T_i$ then no optimal solution assigns job $j$ to $i$, and hence we can remove all such edges from graph $G$.

We model the above matching problem by the following natural linear programming relaxation $LP_{ga}$ to prove Theorem 3.2.1. Observe that we do not place time constraints for all machines but a subset $M' \subseteq M$ which is initialized to $M$. We have a variable $x_e$ for each $e = ij$ denoting whether job $j$ is assigned to machine $i$.

$$
\begin{aligned}
\text{minimize} \quad & \sum_{e=(i,j) \in E} c_{ij}\, x_{ij} & & & \\
\text{subject to} \quad & \sum_{e \in \delta(j)} x_e & = & 1 & \forall\, j \in J \\
& \sum_{e \in \delta(i)} p_e x_e & \leq & T_i & \forall\, i \in M' \\
& x_e & \geq & 0 & \forall\, e \in E
\end{aligned}
$$

### *3.2.2 Characterization of Extreme Point Solutions*

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

**Lemma 3.2.2** *Let $x$ be an extreme point solution to the linear program $LP_{ga}$ such that $0 < x_e < 1$ for each edge $e$. Then there exist $J' \subseteq J$ and $M'' \subseteq M'$ such that*

(i) $\sum_{e \in \delta(j)} x_e = 1$ *for each* $j \in J'$ *and* $\sum_{e \in \delta(i)} p_e x_e = T_i$ *for each* $i \in M''$.

(ii) *The constraints corresponding to $J'$ and $M''$ are linearly independent.*

(iii) $|J'| + |M''| = |E(G)|$

### *3.2.3 Iterative Algorithm*

We present a simple iterative procedure which returns an assignment of optimal cost in Figure 3.2. Observe that the iterative procedure generalizes the iterative procedure for bipartite matching in Section 3.1. The bipartite graph $F$ with vertex set in $M \cup J$ returns the assignment found by the algorithm.

This procedure demonstrates our first example of the iterative relaxation method in Step (ii)(c). Here in addition to the usual step of picking an integral element in the solution, we identify carefully chosen constraints to relax or remove. The choice is dictated by ensuring that the removal will allow us to argue that the final integral solution does not have too much violation; at the same time, we need to ensure that in the absence of an integral element, such a constraint can always be found to be removed. The crux of the relaxation method is to find the right relaxation condition that balances this trade-off nicely.

**Iterative Generalized Assignment Algorithm**

(i) Initialization $E(F) \leftarrow \emptyset$, $M' \leftarrow M$.

(ii) While $J \neq \emptyset$ do

  (a) Find an optimal extreme point solution $x$ to $LP_{ga}$ and remove every variable with $x_{ij} = 0$.

  (b) If there is a variable with $x_{ij} = 1$, then update $F \leftarrow F \cup \{ij\}$, $J \leftarrow J \setminus \{j\}$, $T_i \leftarrow T_i - p_{ij}$.

  (c) **(Relaxation)** If there is a machine $i$ with $d(i) = 1$, or a machine $i$ with $d(i) = 2$ and $\sum_{j \in J} x_{ij} \geq 1$, then update $M' \leftarrow M' \setminus \{i\}$.

(iii) Return $F$.

Fig. 3.2. The Generalized Assignment Algorithm

### *3.2.4 Correctness and Near-Optimality*

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

**Lemma 3.2.3** *Consider any extreme point solution $x$ to $LP_{ga}$. One of the following must hold.*

(i) *There exists an edge $e \in E$ with $x_e \in \{0, 1\}$.*
(ii) *There exists a machine $i \in M'$ with $d(i) = 1$, or $d(i) = 2$ and $\sum_{j \in J} x_{ij} \geq 1$.*

*Proof* Suppose for sake of contradiction that both the conditions do not hold. By Step (ii)(a) and Step (ii)(b), we have $0 < x_e < 1$ for each edge $e$. Each job $j$ has degree at least two since $\sum_{e \in \delta(j)} x_e = 1$ and there is no edge with $x_e = 1$ by Step (ii)(b). Moreover, each machine in $M'$ has degree at least two, because the constraints for machines with degree one have been removed in Step (ii)(c). From Lemma 3.2.2 we have that $|E| = |J'| + |M''|$. This implies that:

$$|J'| + |M''| = |E| \geq \frac{\sum_{j \in J} d(j) + \sum_{i \in M'} d(i)}{2} \geq |J| + |M'| \geq |J'| + |M''|,$$

and hence all inequalities must hold as equalities. The first inequality implies that each machine $i \in M \setminus M'$ has degree zero; the second inequality implies that each job $j \in J'$ and each machine $i \in M''$ have degree exactly two; the last inequality implies that $J = J'$ and $M' = M''$. Therefore, $G$ is a union of cycles, with vertices in $J' \cup M''$ (tight constraints). Consider any cycle $C$. The total number of jobs in $C$ is exactly equal to the total number of machines in $C$. Therefore, since each job $j \in J'$ has $\sum_{i \in M''} x_{ij} = 1$, there must be a machine $i$ with $\sum_{j \in J'} x_{ij} \geq 1$. Hence, this machine $i$ has degree two and $\sum_{j \in J'} x_{ij} \geq 1$, contradicting that Step (ii)(c) cannot be applied. □

We now prove Theorem 3.2.1 by a simple inductive argument.

**Proof of Theorem 3.2.1:** 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 $LP_{ga}$ is at most the cost of the initial linear programming solution. This can be shown 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 (ii)(b) 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. If we remove a constraint in Step (ii)(c), then the cost of $F$ remains the same, while the cost of the current linear program can only decrease. Hence, the claim holds in this case as well. Thus, finally when $F$ is a feasible assignment, by induction, the cost of assignment given by $F$ is at most the cost of the initial linear programming solution.

Now, we show that machine $i$ is used at most $2T_i$ units for each $i$. 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 $F$. The proof of the claim follows by a simple inductive argument as in the above inductive argument for costs. Now consider when the machine $i$ is removed from $M'$. There are two possibilities. If there is only one job $j$ in machine $i$, then the total processing time at machine $i$ is at most $T_i + p_{ij} \leq 2T_i$, where the inequality holds because of the pruning step (where we deleted edges assigning a job to machine $i$ if its processing time exceeded $T_i$). If there are two jobs $j_1$ and $j_2$ in machine $i$ then let $x$ denote the linear programming solution when the constraint for machine $i$ is removed. The total processing time at machine $i$ at most

$$\begin{aligned} & T_i(F) + p_{ij_1} + p_{ij_2} \\ \leq\ & T_i - x_{ij_1}p_{ij_1} - x_{ij_2}p_{ij_2} + p_{ij_1} + p_{ij_2} \\ \leq\ & T_i + (1 - x_{ij_1})p_{ij_1} + (1 - x_{ij_2})p_{ij_2} \\ \leq\ & T_i + (2 - x_{ij_1} - x_{ij_2})T_i \\ \leq\ & 2T_i, \end{aligned}$$

because $p_{ij_1}, p_{ij_2} \leq T_i$ again by the pruning step and $x_{ij_1} + x_{ij_2} \geq 1$ by Step (ii)(c). This completes the proof of Theorem 3.2.1. ■

## 3.3 Maximum Budgeted Allocation

In this section we consider the maximum budgeted allocation problem, which is similar to the general assignment problem but is a maximization problem instead of a minimization problem. There are a set $Q$ of indivisible items and a set $A$ of agents. Each agent $i \in A$ is willing to pay a maximum of $b_{ij}$ dollars for item $j \in Q$, but has a maximum budget $B_i$ on total spending. The maximum budgeted allocation problem is to allocate items to agents to maximize revenue. The main result of this section is the following theorem by Chakrabarty and Goel [22].

**Theorem 3.3.1** *There is a 4/3-approximation algorithm for the maximum budgeted allocation problem.*

### *3.3.1 Linear Programming Relaxation*

This problem can be formulated as an integer linear programming, in which there is a variable $x_{ij}$ for agent $i$ and item $j$ to indicate whether item $j$ is assigned to agent $i$.

$$\begin{aligned} \text{maximize} \quad & \sum_{i \in A} \min(B_i, \sum_{j \in Q} b_{ij} x_{ij}) \\ \text{subject to} \quad & \sum_{i \in A} x_{ij} \leq 1 && \forall\, j \in Q \\ & x_{ij} \in \{0, 1\} && \forall\, i \in A, j \in Q \end{aligned}$$

The constraints require that each item is allocated to at most one agent. The objective function can be rewritten as a linear function by adding auxiliary constraints. Thus we obtain a linear programming relaxation by relaxing the integrality constraints. Observe that by appropriate scaling there is always an optimal fractional solution in which $\sum_{j \in Q} b_{ij} x_{ij} \leq B_i$ for all $i \in A$. Henceforth we consider the following equivalent linear programming relaxation, denoted by $LP_{mba}$, which is similar to that of the general assignment problem. Throughout this section we assume without loss of generality that $b_{ij} \leq B_i$ for each $j \in Q$ (for otherwise, we can reset $b_{ij}$ to $B_i$).

$$
\begin{array}{lrcll}
\text{maximize} & \displaystyle\sum_{i \in A, j \in Q} b_{ij} x_{ij} & & & \\
\text{subject to} & \displaystyle\sum_{j \in Q} b_{ij} x_{ij} & \leq & B_i & \forall\, i \in A \\
& \displaystyle\sum_{i \in A} x_{ij} & \leq & 1 & \forall\, j \in Q \\
& x_{ij} & \geq & 0 & \forall\, i \in A, j \in Q
\end{array}
$$

### *3.3.2 Characterization of Extreme Point Solutions*

Given a fractional solution $x$ to $LP_{mba}$, we construct a bipartite graph $G_x = (A \cup Q, E)$ with $ij \in E$ if and only if $x_{ij} > 0$. The following lemma is a direct application of the Rank Lemma 1.2.3.

**Lemma 3.3.2** *Given any extreme point solution $x$ to $LP_{mba}$ such that $x_{ij} > 0$ for each $i \in A, j \in Q$ there exist $A' \subseteq A, Q' \subseteq Q$ such that*

(i) $\sum_{j \in Q} b_{ij} x_{ij} = B_i$ *for each* $i \in A'$ *and* $\sum_{i \in A} x_{ij} = 1$ *for each* $j \in Q'$.
(ii) *The corresponding row vectors in $A'$ and $Q'$ are linearly independent.*
(iii) $|E| = |A'| + |Q'|$.

Similar to the bipartite matching problem and the generalized assignment problem, the above lemma implies that each connected component of $G_x$ has at most one cycle. With an additional cycle elimination argument that shifts the values of the solution along such a cycle without decreasing the objective value, one can prove that $G_x$ is a forest (see exercises). We call an item a *leaf item* if it is a leaf in $G_x$. We call an agent $i$ a *leaf agent* if all but one neighbor of $i$ are leaf items ($i$ itself may not be a leaf). Every tree in the forest has at least one leaf item and one leaf agent. Since $x$ is an extreme point solution and $G_x$ is a forest, there is at most one non-tight constraint in each component. For an agent that is a leaf of a tree, as its bid to the item is at most the agent's budget, such an agent must be non-tight in a tree with at least two agents (if such an agent is tight then the component has only this agent and this item). It follows that there is at least one

tight leaf agent in a tree with at least two agents. The following lemma summarizes the properties of the extreme point solutions that we will use.

**Lemma 3.3.3** *[22] Given any extreme point solution $x$ to $LP_{mba}$:*

(i) *The graph $G_x$ is a forest;*

(ii) *There is at least one leaf agent in each component of $G_x$.*

(iii) *There is at least one tight leaf agent in a component of $G_x$ with at least two agents.*

### *3.3.3 An Iterative 2-Approximation Algorithm*

We first present a simple iterative 2-approximation algorithm for the problem. In the following $N(i)$ denotes the set of neighbors (items) of agent $i$ in $G_x$.

**Iterative 2-Approximation Algorithm for Maximum Budgeted Allocation**

While $Q \neq \emptyset$ do

(i) Find an optimal extreme point solution $x$ to $LP_{mba}$. Remove all edges with $x_{ij} = 0$.

(ii) Pick a leaf agent $i$. Let $L$ be the set of leaf items in $N(i)$. Assign each item $l \in L$ to $i$ and then remove $L$. Modify $B_i' := B_i - \sum_{l \in L} b_{il} x_{il}$.

(a) If $i$ has a non-leaf item $j$, modify the bid of $i$ on $j$ to be $b_{ij} \leftarrow \min(b_{ij}, B_i')$.

Fig. 3.3. The Maximum Budgeted Allocation 2-Approximation Algorithm

We now prove that the above algorithm is a 2-approximation algorithm. Consider an arbitrary iteration. The increase of the integral solution is at least $\sum_{l \in L} b_{il} x_{il}$. Note that $x$ restricted to the remaining edges is a feasible solution to the residual problem. If $b'_{ij} = b_{ij}$, then the decrease of the fractional solution is at most $\sum_{l \in L} b_{il} x_{il}$. Let $b'_{ij} = \min(b_{ij}, B'_i)$ denote the new bid of agent $i$ to item $j$. Otherwise, if $b'_{ij} = B'_i$, then the decrease of the fractional solution is at most

$$\sum_{l \in L} b_{il} x_{il} + (b_{ij} - b'_{ij}) x_{ij} \leq \sum_{l \in L} b_{il} x_{il} + (B_i - B'_i) = 2 \sum_{l \in L} b_{il} x_{il},$$

where the first inequality follows because $b_{ij} \leq B_i$ and $b'_{ij} = B'_i$ and $x_{ij} \leq 1$. Hence, in either case, the increase of the integral solution is at least half the decrease of the fractional solution in each iteration. It follows by an inductive argument that the final integral solution is at least half the initial fractional solution.

### *3.3.4 An Iterative $\frac{4}{3}$-Approximation Algorithm*

Here we present an improved iterative approximation algorithm for the problem. In Step (ii)(a) of the algorithm in Figure 3.3, the bid of agent $i$ on item $j$ is decreased by the amount collected from the leaf items, and this is the bottleneck of achieving a better approximation algorithm. The new idea is to keep the new bid slightly higher. The intuition is that there is some "surplus" from the amount collected from the leaf items, since they are collected with "factor one" while we only require an approximate solution. The improved algorithm is presented in Figure 3.4.

**Iterative $\frac{4}{3}$-Approximation Algorithm for Maximum Budgeted Allocation**

While $Q \neq \emptyset$ do

(i) Find an optimal extreme point solution $x$ to $LP_{mba}$. Remove all edges with $x_{ij} = 0$.

(ii) If there is a leaf agent $i$ with all the items in $N(i)$ being leaf items, then assign each item $l \in N(i)$ to $i$ and remove $N(i)$ and $i$.

(iii) Pick a tight leaf agent $i$. Let $L$ be the set of leaf items in $N(i)$, and $j$ be the unique non-leaf item in $N(i)$. Assign each item $l \in L$ to $i$ and then remove $L$. Modify both

$$B_i, b_{ij} \leftarrow \max(0, \frac{4b_{ij}x_{ij} - B_i}{3x_{ij}})$$

Fig. 3.4. The Maximum Budgeted Allocation 4/3-Approximation Algorithm

### *3.3.5 Correctness and Performance Guarantee*

Lemma 3.3.3 guarantees that the algorithm will terminate successfully. Therefore, it remains to prove that the returned integral solution is at least $\frac{3}{4}$ of the initial fractional solution. As in the proof of the 2-approximation algorithm, we apply the following lemma inductively to complete the proof.

**Lemma 3.3.4** *In any iteration the increase of the integral solution is at least $\frac{3}{4}$ of the decrease of the fractional solution.*

*Proof* Consider any iteration and first suppose that Step (ii) is applied on a leaf agent $i$ in this iteration. Let $x$ denote the optimal fractional solution at the end of the iteration. Note that $B_i$ is modified at most once throughout the algorithm. Suppose $B_i$ has not been modified before this iteration. Then the increase of the integral solution is at least $\sum_{l \in N(i)} b_{il}x_{il}$. Since the current solution $x$ restricted to the remaining edges is a feasible solution in the residual problem, the decrease of the fractional solution is at most

$\sum_{l \in N(i)} b_{il} x_{il}$, and thus the lemma holds. Now suppose that $B_i$ has been modified before this iteration. In this case let $j$ be the unique neighbor of $i$. Let $y$ denote the fractional solution when $B_i$ was modified. Let $B_i$ denote the original budget, $b_{ij}$ denote the original bid and $b'_{ij}$ denote the current bid. The decrease in the fractional solution in the current step is at most its current bid

$$b'_{ij} = \frac{4b_{ij}y_{ij} - B_i}{3y_{ij}}.$$

However, from the iteration when its budget was modified, the unspent budget left on agent $i$ is at least $b_{ij}y_{ij}$ and hence the increase of the integral solution is at least

$$b_{ij}y_{ij} \geq \frac{3}{4} \cdot \frac{4b_{ij}y_{ij} - B_i}{3y_{ij}}.$$

Therefore the lemma holds if Step (ii) is applied.

Now consider the case when Step (iii) is applied on a tight leaf agent $i$. Let $j$ be the unique non-leaf item in $N(i)$. Let $b'_{ij}$ denote the new bid and $B'_i$ denote the new budget. Since the current solution $x$ restricted to the remaining edges is a feasible solution in the residual problem, the decrease of the fractional solution is at most

$$\sum_{l \in L} b_{il}x_{il} + (b_{ij} - b'_{ij})x_{ij}.$$

On the other hand, the increase of the integral solution is at least

$$\sum_{l \in L} b_{il}x_{il}.$$

To complete the proof, we need to show that the increase of the integral solution is at least 3/4 of the decrease of the fractional solution, which is equivalent to

$$\frac{1}{3}\sum_{l \in L} b_{il}x_{il} \geq (b_{ij} - b'_{ij})x_{ij}.$$

Since $i$ is a tight agent, it is equivalent to showing that

$$\frac{1}{3}(B_i - b_{ij}x_{ij}) \geq (b_{ij} - b'_{ij})x_{ij}.$$

This holds for $b'_{ij} = \max\{0, \frac{4b_{ij}x_{ij} - B_i}{3x_{ij}}\} = 0$, because in this case $4b_{ij}x_{ij} \leq B_i$, and thus the above inequality holds. Otherwise, we have

$$b_{ij}x_{ij} - b'_{ij}x_{ij} = b_{ij}x_{ij} - \frac{4b_{ij}x_{ij} - B_i}{3x_{ij}} \cdot x_{ij} = \frac{B_i - b_{ij}x_{ij}}{3}.$$

This completes the proof of the lemma, and thus the proof of Theorem 3.3.1. □

## 3.4 Vertex Cover in Bipartite Graphs

In this section we study the vertex cover problem in bipartite graphs. Given a graph $G = (V, E)$ with cost function $c : V \to \mathbb{R}_+$ the vertex cover problem asks for a set of vertices $V'$ such that $e \cap V' \neq \emptyset$ for each $e \in E$ and $c(V') = \sum_{v \in V'} c_v$ is minimized. We restrict our attention to bipartite graphs, i.e, $V = V_1 \cup V_2$.

### *3.4.1 Linear Programming Relaxation*

We first give the following natural linear programming relaxation $LP_{bvc}(G)$ for the vertex cover problem in bipartite graphs. We have a variable $x_v$ for each vertex $v$ denoting its inclusion in the solution.

$$\begin{array}{lrcll} \text{minimize} & \displaystyle\sum_{v \in V} c_v\, x_v & & & \\ \text{subject to} & x_u + x_v & \geq & 1 & \forall\, e = \{u, v\} \in E \\ & x_v & \geq & 0 & \forall\, v \in V \end{array}$$

As in the previous two sections, the linear program $LP_{bvc}(G)$ 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 by Theorem 2.1.6.

We prove the following theorem by an iterative algorithm.

**Theorem 3.4.1** *Given any cost function $c$ there exists an integral vertex cover $U$ such that $c(U) \leq c \cdot x$ where $x$ is an optimal solution to $LP_{bvc}(G)$.*

As before, our proof of Theorem 3.4.1 as a corollary implies the following theorem.

**Theorem 3.4.2** *The linear programming formulation $LP_{bvc}(G)$ is integral.*

### *3.4.2 Characterization of Extreme Point Solutions*

Before we prove Theorem 3.4.1 we give a characterization of extreme points of $LP_{bvc}$. For a set $W \subseteq V$, let $\chi(W)$ denote the vector in $\mathbb{R}^{|V|}$: the vector has an 1 corresponding to each vertex $v \in W$, and 0 otherwise. This vector is called the *characteristic vector* of $W$, and is denoted by $\chi(W)$. In the following lemma, which follows by a direct application of the Rank Lemma 1.2.3, we characterize an extreme point solution by a set of tight independent constraints.

**Lemma 3.4.3** *Given any extreme point $x$ to $LP_{bvc}(G)$ with $x_v > 0$ for each $v \in V$ there exists $F \subseteq E$ such that*

(i) $x_u + x_v = 1$ *for each* $e = \{u, v\} \in F$.
(ii) *The vectors* $\{\chi(\{u, v\}) : \{u, v\} \in F\}$ *are linearly independent.*
(iii) $|V| = |F|$.

### *3.4.3 Iterative Algorithm*

We now give the algorithm which constructs an integral vertex cover of cost at most the optimal solution to $LP_{bvc}(G)$ proving Theorem 3.4.1. The algorithm is a simple iterative procedure and shown in Figure 3.5.

**Iterative Bipartite Vertex Cover Algorithm**

(i) Initialization $U \leftarrow \emptyset$.
(ii) While $V(G) \neq \emptyset$ do
  (a) Find an optimal extreme point solution $x$ to $LP_{bvc}(G)$ and remove every vertex $v$ with $x_v = 0$ and $d_E(v) = 0$ from $G$.
  (b) If there is a vertex $v \in V$ such that $x_v = 1$ then update $U \leftarrow U \cup \{v\}$, $V(G) \leftarrow V(G) \setminus \{v\}$ and $E(G) \leftarrow E(G) \setminus \delta(v)$.
(iii) Return $U$.

Fig. 3.5. Bipartite Vertex Cover Algorithm

### *3.4.4 Correctness and Optimality*

Following the approach we used for bipartite matching, we prove the correctness of the algorithm in two steps. First, we show that the algorithm returns a vertex cover of optimal cost if the algorithm *always* finds a vertex $v$ with $x_v = 0$ in Step (ii)(a) or a vertex $v$ with $x_v = 1$ in Step (ii)(b) . In the second part, we show that the algorithm will always find such an vertex completing the proof.

**Claim 3.4.4** *If the algorithm, in every iteration, finds a vertex* $v$ *with* $x_v = 0$ *and* $d_E(v) = 0$ *in Step* (ii)(a) *or a vertex* $v$ *with* $x_v = 1$ *in Step* (ii)(b)*, then it returns a vertex cover* $U$ *of cost at most the optimal solution to* $LP_{bvc}(G)$.

The proof of this claim is identical to the proof of Claim 3.1.4. We leave the details to the reader. We now complete the proof of Theorem 3.4.1 by showing that we can always find a vertex $v$ with $x_v = 0$ and $d_E(v) = 0$, or a vertex with $x_v = 1$. The proof of the following lemma crucially uses the characterization of extreme point solutions given in Lemma 3.4.3.

**Lemma 3.4.5** *Given any extreme point solution* $x$ *to* $LP_{bvc}(G)$ *there must exist a vertex* $v$ *with* $x_v = 0$ *and* $d_E(v) = 0$*, or a vertex with* $x_v = 1$.

*Proof* Suppose for sake of contradiction that $x_v < 1$ and $x_v = 0$ implies that $d_E(v) \geq 1$ for each vertex $v \in V$. This implies that there is no vertex with $x_v = 0$. This follows from the fact that any neighbor $u$ of $v$ must have $x_u = 1$ since $x_u + x_v \geq 1$. Lemma 3.4.3 implies that there exists $F \subseteq E$ such that constraints corresponding to $F$ are tight and $|F| = |V|$.

**Claim 3.4.6** *$F$ is acyclic.*

*Proof* Suppose for sake of contradiction $C \subseteq F$ is a cycle. Since $G$ is bipartite $C$ is an even cycle. Then $C$ is the disjoint union of two matchings $M_1$ and $M_2$. But then the sum of the constraints corresponding to edges in $M_1$ equals the sum of the constraints corresponding to edges in $M_2$, contradicting the independence of constraints for edges in $F$. □

Since $F$ is acyclic, we must have $|F| \leq |V| - 1$, a contradiction. □

Thus, by Lemma 3.4.5, the algorithm in Figure 3.5 returns a vertex cover that costs at most the cost of the optimal solution to the linear program $LP_{bvc}(G)$, proving Theorem 3.4.1.

## 3.5 Vertex Cover and Matchings: Duality

In this section we prove the following min-max theorem between the size of minimum vertex covers and maximum matchings in bipartite graphs using the integrality proofs we have seen in previous sections.

**Theorem 3.5.1** *Given an unweighted bipartite graph $G = (V, E)$ we have*

$$\max\{|M| : M \text{ is a matching}\} = \min\{|U| : U \text{ is a vertex cover}\}$$

*Proof* Let $M^*$ be the maximum weight matching returned by the Iterative Algorithm in Figure 3.1 when the weight function $w_e = 1$ for all $e \in E$. Also, let $U^*$ denote the minimum cost vertex cover returned by the iterative algorithm in Figure 3.5 when the cost function $c_v = 1$ for each $v \in V$. From Theorem 3.1.1 and Theorem 3.4.1 we have that $M^*$ is an optimal solution to $LP_{bm}(G)$ and $U^*$ is an optimal solution to $LP_{bvc}(G)$. But $LP_{bm}(G)$ and $LP_{bvc}(G)$ are duals of each other when $w$ and $c$ are uniformly one (see Section 2.1.4). By the strong duality theorem (Theorem 2.1.9) both the linear programs must have optimal solutions of equal value:

$$|M^*| = \max\{|M| : M \text{ is a matching}\} = \min\{|U| : U \text{ is a vertex cover}\} = |U^*|.$$

□

A weighted generalization of the above theorem is true. It can be similarly shown using duality and the fact that the corresponding linear programming problems have integral optima. For more details, see Chapter 18 of Schrijver's book [130].

## Notes

The bipartite perfect matching problem (also known as the Assignment problem) is one of the oldest problems in Combinatorial Optimization (see e.g. [128]). The first proof of polynomial solvability of maximum weight bipartite matching via the Hungarian method was given by Egervary [44], sharpened by Kuhn [94] and shown efficient by Munkres [107]. The unweighted case was addressed earlier by Frobenius [52]. Birkhoff [15] was the first to show that the extreme points of the bipartite matching polyhedron (defined by the so-called "doubly stochastic" constraint matrix) are integral.

The generalized assignment problem was first studied by Shmoys and Tardos [131] following up on a bi-criteria approximation algorithms for the problem due to Lin and Vitter [98]. Trick [136] studied the version minimizing the weighted sum of cost and makespan of the underlying scheduling problem, while Lenstra, Shmoys and Tardos [97] study the makespan problem and give the 2-approximation that was generalized in the subsequent work of Shmoys and Tardos [131]

The maximum budgeted allocation problem is NP-hard. The first approximation algorithm is a $(1 + \sqrt{5})/2$-approximation algorithm given by Garg, Kumar and Pandit [60]. The approximation ratio was subsequently improved to $e/(e-1)$ by Andelman and Mansour [2], and $3/2$ by Azar et al. [6]. The result presented in this chapter is by Chakrabarty and Goel [22]. The same result is also obtained independently by Srinivasan [135] by a dependent rounding algorithm.

The bipartite minimum vertex cover problem is the linear programming dual to the maximum matching problem. The min-max theorem relating them is due to König [90]. The Hungarian method and LP duality can be used to show the integrality of the weighted vertex cover problem. An alternate approach is to use the total unimodularity (see e.g., the book by Nemhauser and Wolsey [113]) of the constraint matrix which is the edge-node incidence matrix of the bipartite graph, which is also an example of a Network Matrix that we will study in Chapter 8.

## Exercises

3.1 Given a bipartite graph $G = (V, E)$ and a cost function $c : E \rightarrow \mathcal{R}$, a perfect matching is a subgraph with degree exactly one at any node.

(a) Write a linear programming formulation for the minimum cost perfect matching problem.

(b) Give an iterative proof of integrality of the linear programming formulation for the minimum cost perfect matching problem. (Hint: Adapt the proof for maximum weight matchings.)

3.2 Show that $LP_{bm}(G)$ and $LP_{bvc}(G)$ are *not* integral when $G$ is not bipartite.

3.3 Generalize the methods given in this chapter for maximum weight matchings and

minimum weight vertex covers in bipartite graphs to the case when the right hand side of the LP constraints are positive integers (rather than all 1's as we considered). Note that the two resulting integrality proofs, along with strong duality, will imply the general min-max relation between maximum-weight matchings with arbitrary vertex degree bounds and minimum-cost vertex cover with arbitrary coverage requirements at the edges.

3.4 Given a set of intervals $[a_i, b_i]$ for each $1 \leq i \leq n$ and a weight function $w$ on intervals, the *maximum weight k-interval packing* problem asks for a subset $J$ of intervals of maximum weight such that there are at most $k$ intervals in $J$ at any point on the line.

(a) Formulate a linear program for the maximum weight $k$-interval packing problem.

(b) Show that only $n - 1$ *point* constraints need to be imposed apart from the bound constraints.

(c) Show that the linear program is integral.

3.5 Can you solve the maximum weight $k$-interval packing problem for intervals on a tree rather than a path (as was the case in the previous problem) using the same approach? If you can, give an integral description for the problem. If not, argue why natural formulations are not integral.

Consider the case when all the intervals in a tree are monotone, i.e., they go from a point in a rooted version of the tree to its ancestor (thus there are no intervals whose endpoints are two incomparable points in the tree). Can you give an integral LP formulation in this case using the approach of the previous problem?

3.6 Prove Lemma 3.3.3 starting from Lemma 3.3.2.

3.7 Construct an example for $LP_{mba}$ of the maximum budgeted allocation problem with integrality gap $\frac{4}{3}$.

# 4
# Spanning Trees

In this chapter we will study the spanning tree problem in undirected graphs. First, we will study an exact linear programming formulation and show its integrality using the iterative method. To do this, we will introduce the *uncrossing method*, which is a very powerful technique in combinatorial optimization. The uncrossing method will play a crucial role in the proof and will occur at numerous places in later chapters. We will show two different iterative algorithms for the spanning tree problem, each using a different choice of 1-elements to pick in the solution. For the second iterative algorithm, we show three different correctness proofs for the existence of an 1-element in an extreme point solution: a global counting argument, a local integral token counting argument and a local fractional token counting argument. These token counting arguments will be used in many proofs in later chapters.

We then address the degree-bounded minimum-cost spanning tree problem. We show how the methods developed for the exact characterization of the spanning tree polyhedron are useful in designing approximation algorithms for this NP-hard problem. We give two additive approximation algorithm: the first follows the first approach for spanning trees and naturally generalizes to give a simple proof of the additive two approximation result of Goemans [62]; the second follows the second approach for spanning trees and uses the local fractional token counting argument to provide a very simple proof of the additive one approximation result of Singh and Lau [134].

## 4.1 Minimum Spanning Trees

In an instance of the minimum spanning tree (MST) problem we are given an undirected graph $G = (V, E)$, edge costs given as $c : E \to \mathbb{R}$, and the task is to find a spanning tree of minimum total edge cost.

### *4.1.1 Linear Programming Relaxation*

In this section two formulations of the minimum spanning tree problem are discussed (some more are discussed in the Exercises). A spanning tree is a minimal 1-edge-connected subgraph. Thus, a natural formulation (sometimes called the undirected LP or the cut LP), is to require that every pair of vertices has a path connecting them. Or equivalently, we can require that there is at least one edge crossing each proper subset of vertices.

$$\begin{array}{lrcll} \text{minimize} & \displaystyle\sum_{e\in E} c_e\, x_e & & & \\ \text{subject to} & x(\delta(S)) & \geq & 1 & \forall\ S \subset V \\ & x_e & \geq & 0 & \forall\ e \in E \end{array}$$

There is a variable $x_e$ corresponding to each edge $e$, to indicate that whether $e$ is included in the spanning tree, and $c_e$ denotes the cost of $e$. For a set $F$ of edges, the shorthand $x(F)$ is used to denote $\sum_{e\in F} x_e$. Recall that $\delta(S)$ is the set of edges with exactly one endpoint in $S$.

There are exponentially many constraints in the undirected LP. However, from Theorem 2.1.8 in Chapter 1, the undirected LP can still be solved in polynomial time if a polynomial time separation oracle can be constructed. Constructing such a polynomial-time oracle is equivalent to finding a cut of total capacity less than one in the graph with capacities $x$, and can be accomplished using a global minimum-cut algorithm. (In detail, given a solution to the above linear program, one needs to determine whether it is a feasible solution, and if not, provides a violating inequality. A polynomial time separation oracle for the undirected LP is easy to construct. Given a fractional solution, check first if every variable is nonnegative. Then, for every pair of vertices, check if the maximum flow between them is at least 1. If this condition is satisfied for every pair, then clearly the given fractional solution is a feasible solution. On the other hand, if for some pair this condition is not satisfied, by the max-flow min-cut theorem, there is a set $S$ with $x(\delta(S)) < 1$, and such a set can be found in polynomial time.) Unfortunately, the undirected LP is not an exact formulation of the minimum spanning tree problem as shown in Figure 4.1.

Another formulation is the subtour elimination LP which is related to the study of the traveling salesman problem (TSP). For $S \subseteq V$, define $E(S)$ to be the set of edges with both endpoints in $S$. For a spanning tree, there are at most $|S| - 1$ edges in $E(S)$, where $|S|$ denotes the number of vertices in $S$. Insisting on this for every set by using the constraint (4.2) eliminates all the potential subtours that can be formed in the LP solution: this is how the formulation gets its name.

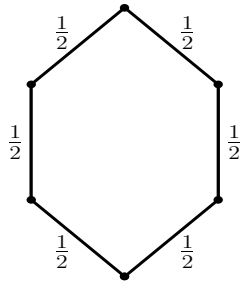


Fig. 4.1. Consider a cycle of $n$ vertices, where every edge has the same cost, say 1. Any spanning tree requires at least $n-1$ edges, and thus has cost at least $n-1$. However, by setting $x_e = 1/2$ for each edge, it can be easily checked that this is a feasible solution to the undirected LP, and has total cost only $n/2$.

$$\text{minimize} \quad \sum_{e \in E} c_e \, x_e \tag{4.1}$$

$$\text{subject to} \quad x(E(S)) \leq |S| - 1 \qquad \forall \, \emptyset \neq S \subset V \tag{4.2}$$

$$x(E(V)) = |V| - 1 \tag{4.3}$$

$$x_e \geq 0 \qquad \forall \, e \in E \tag{4.4}$$

In the next section we will give an iterative algorithm which will prove that the subtour LP is integral.

**Theorem 4.1.1** *Every extreme point solution to the subtour LP is integral and corresponds to the characteristic vector of a spanning tree.*

Before we give the iterative algorithm and proof of Theorem 4.1.1, we show that one can optimize over the subtour LP in polynomial time. We show this by giving a polynomial time separation oracle for the constraints in subtour LP. Polynomial time solvability now follows from Theorem 2.1.8.

**Theorem 4.1.2** *There is a polynomial time separation oracle for the subtour LP.*

*Proof* The separation oracle, given a fractional solution $x$, 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 $S$ is equivalent to checking if $\min_S\{|S|-1-x(E(S))\} < 0$. Using $x(E(V)) = |V| - 1$ we obtain that it is enough to check $\min_S\{|S| - 1 + x(E(V)) - x(E(S))\} < |V| - 1\}$ or equivalently if $\min_S\{|S| + x(E(V)) - x(E(S))\} < |V|\}$.

We set up a min-cut problem in a new digraph $D$ with a new source $s$ and a new sink $t$. We also have one node in the digraph per edge $e$ in the support (i.e. with $x_e > 0$) and a node per vertex of $G$. The source $s$ has an arc to every edge $e$ (i.e. the node in $D$ corresponding to $e$ in $G$) with capacity $x_e$. For every edge $e = i, j$ in $G$, its corresponding

node in $D$ has two arcs of infinite capacity, one to each of the nodes in $D$ corresponding to the vertices $i$ and $j$ in $G$. Finally, every node $i$ in $D$ has an arc of unit capacity to the sink $t$. To find a violated cut, we check if the min $s-t$ cut is smaller than $|V|$.

Suppose there is a violated set $S$ with $x(E(S)) > |S| - 1$. Then consider the cut formed by including on the side of $s$, all the nodes of $D$ corresponding to edges in $E(S)$ as well as all the nodes of $D$ corresponding to the vertices of $S$. The set of arcs coming out of this cut are those coming out of the vertices of $S$ and hence have capacity $|S|$. All edges not in $E(S)$, namely in $E(V) - E(S)$, must now have their incoming arc from $s$ in the cut for a total capacity contribution of $x(E(V)) - x(E(S))$. Thus a violated set will correspond to a cut with value less than $|V|$.

Conversely, suppose the min-cut solution returns one of value less than $|V|$: we show how to extract a violated set from it. Since every node in $D$ corresponding to an edge $e$ of $G$ has both its outgoing arcs with infinite capacity, if such a node (say $e = i, j$) is in the $s$-side of the min cut, then both its successors (i.e. both $i$ and $j$) must also be in the $s$-side of the minimum cut. Similarly, if we take all the vertices of $G$ in the $s$-side of the min-cut, all edges of $G$ which do not have both their endpoints in this set will have to lie on the $t$-side of this cut. If the min-cut found has the set $S'$ in the $s$-side of the cut, the capacity of the cut is precisely $|S'|$ (from the unit arcs going from these nodes to $t$) plus the $x$-value of all edges that do not have both end points in $S$, namely $x(E(V)) - x(E(S'))$, as required. If this min-cut value is less than $|V|$, we can see that $S'$ is a violating set. □

### *4.1.2 Characterization of Extreme Point Solutions*

In this subsection, we analyze the extreme point solutions to the subtour LP. Recall that an extreme point solution is the unique solution defined by $n$ linearly independent tight inequalities, where $n$ is the number of variables in the linear program. There are exponentially many inequalities in the subtour LP, and an extreme point solution may satisfy many inequalities as equalities. To analyze an extreme point solution, an important step is to find a "good" 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 $x_e > 0$.

### *4.1.3 Uncrossing Technique*

The uncrossing technique is a powerful technique and we shall use it to find a *good* set of tight inequalities for an extreme point solution in the subtour LP. Let $E(X, Y)$ denotes the set of edges with one endpoint in $X$ and the other endpoint in $Y$, and let $E(X) = E(X, X)$ denote the set of edges of $G$ induced in $X \subseteq V(G)$. For a set $F \subseteq E$, let $\chi(F)$ denote the vector in $\mathbb{R}^{|E|}$ that has an 1 corresponding to each edge $e \in F$, and 0 otherwise. This vector is called the characteristic vector of $F$. The following proposition follows from the supermodularity of $|E(X)|$; see also Proposition 2.3.6.

**Proposition 4.1.3** *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 4.2. □

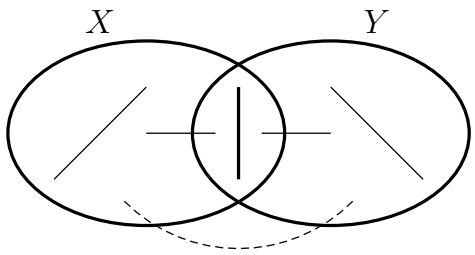


Fig. 4.2. In this example the solid edges are counted exactly once in both the LHS $\chi(E(X)) + \chi(E(Y))$, and the RHS $\chi(E(X \cup Y)) + \chi(E(X \cap Y))$, and the bold edge is counted exactly twice on both sides. The dashed edge, however is counted in the RHS but not in the LHS.

Given an extreme point solution $x$ to the subtour LP, let $\mathcal{F} = \{S \mid x(E(S)) = |S|-1\}$ be the family of tight inequalities for $x$. The following lemma shows that this family is closed under intersection and union.

**Lemma 4.1.4** *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 4.1.3 (or from Proposition 2.3.6). The third inequality follows from the constraints for $S \cap T$ and $S \cup T$ in the subtour LP. The last equality is because $|S| + |T| = |S \cap T| + |S \cup T|$ for any two sets $S, T$. Thus equality must hold everywhere

and we have $x(E(S \cap T)) + x(E(S \cup T)) = |S \cap T| - 1 + |S \cup T| - 1$. Hence, 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 4.1.3 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 $\{\chi(E(S)) \mid S \in \mathcal{F}\}$. Call two sets $X, Y$ *intersecting* if $X \cap Y$, $X - Y$ and $Y - X$ are nonempty. A family of sets is *laminar* if no two sets are intersecting. The following lemma says that an extreme point solution is characterized by tight inequalities whose corresponding sets form a laminar family. This is a crucial structure theorem on the extreme point solutions for the subtour LP.

**Lemma 4.1.5** *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 $span(\mathcal{L}) \subset span(\mathcal{F})$. 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. $intersect(S, \mathcal{L}) = |\{T \in \mathcal{L} \mid S \text{ and } T \text{ are intersecting}\}|$. Since $span(\mathcal{L}) \subset span(\mathcal{F})$, there exists a set $S$ 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 $S, T \in F$, by Lemma 4.1.4, 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 $intersect(S, \mathcal{L})$, which will be proved next in Proposition 4.1.6. Hence, by the minimality of $intersect(S, \mathcal{L})$, both $S \cap T$ and $S \cup T$ are in $span(\mathcal{L})$. By Lemma 4.1.4, $\chi(E(S)) + \chi(E(T)) = \chi(E(S \cap T)) + \chi(E(S \cup T))$. Since $\chi(E(S \cap T))$ and $\chi(E(S \cup T))$ 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 4.1.6.

**Proposition 4.1.6** *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 $S$. 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 $intersect(S, \mathcal{L})$ □

This completes the proof of Lemma 4.1.5. □

The following proposition about the size of a laminar family will be used to bound the number of variables in an extreme point solution.

**Proposition 4.1.7** *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. □

The following corollary follows immediately from Proposition 4.1.7.

**Corollary 4.1.8** *A laminar family $\mathcal{L}$ over the ground set $V$ (potentially including singletons now) has at most $2|V| - 1$ distinct members.*

### *4.1.4 Leaf-finding Iterative Algorithm*

In this subsection, an iterative procedure to find a minimum spanning tree from an optimal extreme point solution of the subtour LP is presented. The algorithm is shown in Figure 4.3.

**Iterative Leaf-finding MST Algorithm**

(i) Initialization $F \leftarrow \emptyset$.
(ii) While $|V(G)| \geq 2$ do
  (a) Find an optimal extreme point solution $x$ to the subtour LP and remove every edge $e$ with $x_e = 0$ from $G$.
  (b) Find a vertex $v$ with at most one edge $e = uv$ incident on it, then update $F \leftarrow F \cup \{e\}$ and $G \leftarrow G \setminus \{v\}$.
(iii) Return $F$.

Fig. 4.3. Iterative Leaf-finding MST Algorithm

### *4.1.5 Correctness and Optimality of Leaf-finding Algorithm*

First, we prove that the algorithm will terminate in the following lemma.

**Lemma 4.1.9** *For any extreme point solution $x$ to the subtour LP with $x_e > 0$ for every edge $e$, there exists a vertex $v$ with $d(v) = 1$.*

*Proof* Suppose each vertex is of degree at least two. Then $|E| = \frac{1}{2}\sum_{v \in V} d(v) \geq |V|$. On the other hand, since there is no edge $e$ with $x_e = 0$, every tight inequality is of the form $x(E(S)) = |S| - 1$. By Lemma 4.1.5, there are $|\mathcal{L}|$ linearly independent tight constraints of

the form $x(E(S)) = |S| - 1$, where $\mathcal{L}$ is a laminar family with no singleton sets. It follows that $|E| = |\mathcal{L}|$ by the Rank Lemma 1.2.3. By Proposition 4.1.7, $|\mathcal{L}| \leq |V| - 1$ and hence $|E| \leq |V| - 1$, a contradiction. □

Next we show that the returned solution is a minimum spanning tree in the following theorem.

**Theorem 4.1.10** *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 (ii)(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 of the LP (subtracting the constraint $x(E(V - v)) \leq |V| - 2$ from the constraint $x(E(V)) = |V| - 1$). In the iterative leaf-finding algorithm, $e$ is added to the solution $F$ (starting with an initial $F = \emptyset$), and $v$ is removed from the graph. Note that for any spanning tree $T'$ of $G' = G \setminus \{v\}$, we can construct a spanning tree $T = T' \cup \{e\}$ of $G$. 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.

Since $x_e = 1$, the restriction of $x$ to $E(G')$, denoted by $x_{res}$, is a feasible solution to the subtour LP for $G'$. Inductively, the algorithm will return a spanning tree $F'$ of $G'$ of cost at most the optimal value of the subtour LP for $G'$, and hence $c(F') \leq c \cdot x_{res}$. 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 4.1.11** *If $x$ is an optimal extreme point solution to the subtour LP for $G$, then the residual LP solution $x_{res}$, $x$ restricted to $G' = G \setminus v$, remains an optimal extreme point solution to the subtour LP for $G'$. Hence, in the Iterative MST Algorithm we only need to solve the original linear program once and none of the residual linear programs.*

Theorem 4.1.10 also shows that the subtour LP is an exact formulation of the minimum spanning tree problem showing the proof of Theorem 4.1.1.

Alternately, note that the proof of Lemma 4.1.9 already showed that $|E| = n - 1$ and since $x(E) = n - 1$ and $x(e) \leq 1$ for all edges $e \in E$ (by considering the constraint $x(E(S)) = |S| - 1$ for the size-two set $S$ defined by the endpoints of the edge), 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. This gives a direct (non-iterative) proof of integrality of the subtour LP.

## 4.2 Iterative 1-edge-finding Algorithm

In this section, we give another iterative procedure to find a minimum spanning tree from an optimal extreme point solution to the subtour LP is presented. The algorithm is shown in Figure 4.4. A key difference is that to create the residual problem, the chosen edge $e$ is contracted from $G$ to identify its endpoints to result in the graph $G/e$.

**Iterative 1-edge-finding MST Algorithm**

(i) Initialization $F \leftarrow \emptyset$.
(ii) While $|V(G)| \geq 2$ do
  (a) Find an optimal extreme point solution $x$ to the subtour LP and remove every edge $e$ with $x_e = 0$ from $G$.
  (b) Find an edge $e = \{u, v\}$ with $x_e = 1$ and update $F \leftarrow F \cup \{e\}$, $G \leftarrow G/e$.
(iii) Return $F$.

Fig. 4.4. Iterative 1-edge-finding MST Algorithm

### *4.2.1 Correctness and Optimality*

Following the discussion in Subsection 4.1.5 it is enough to show that the algorithm will terminate. An argument similar to one in proof of Theorem 4.1.10 will show that the output of the algorithm is a minimum spanning tree.

**Lemma 4.2.1** *For any extreme point solution $x$ to the subtour LP with $x_e > 0$ for each edge $e$ there exists an edge $f$ such that $x_f = 1$.*

*Proof* We give three alternate proofs of this lemma mainly to illustrate the three types of counting arguments that can be used to accomplish such proofs.

**Proof 1 (Global Counting Argument).** This is the proof style of Lemma 4.1.5 which shows by a global degree counting argument over all (non-leaf) nodes that $|E| \geq |V|$, which contradicts the upper bound of $|\mathcal{L}|$ of $|V| - 1$.

**Proof 2 (Local Token Counting Argument).** By Lemma 4.1.5, there are $|\mathcal{L}|$ linearly independent tight constraints of the form $x(E(S)) = |S| - 1$, and so $|E| = |\mathcal{L}|$. We now show a contradiction to this through a local token counting argument.

We assign one token for each edge $e$ in the support $E$, for a total of $|E|$ tokens. We

will redistribute the tokens so that each set in $\mathcal{L}$ will receive one token and there are some extra tokens left. This implies that $|E| > |L|$, giving us the contradiction. Actually, for the contradiction, we can collect two tokens for each set $S \in \mathcal{L}$. Since $|\mathcal{L}| \geq 1$, this will give the desired extra token and hence the contradiction.

To redistribute the tokens, each edge gives its token to the smallest set containing both of its endpoints. Let $S$ be any set in $\mathcal{L}$ with children $R_1, \ldots, R_k$ ($k$ could be zero). We have

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

and for each $i$,

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

Subtracting, we obtain

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

Let $A = E(S) \setminus (\cup_i E(R_i))$. 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 linear independence of these constraints in $\mathcal{L}$. Moreover, $|A| \neq 1$ as $x(A)$ is an integer but no single edge in it has an integral value. Hence, $|A| \geq 2$ and thus $S$ receives at least two tokens.

**Proof 3 (Local Fractional Token Counting Argument).** This is a slight modification of the previous argument but generalizes nicely to the degree-bounded case. As before, we assign one token for each edge $e$ in the support $E$, for a total of $|E|$ tokens. For each edge $e$, however, we only redistribute $x_e$ *fractional* token to the smallest set containing both the endpoints. Now, we show that each set in $\mathcal{L}$ can collect at least one token, and demonstrate some extra leftover fractional edge tokens as before giving us the contradiction.

Let $S$ be any set in $\mathcal{L}$ with children $R_1, \ldots, R_k$. Following the previous proof, we have that

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

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

where $A = E(S) \setminus (\cup_i E(R_i))$. Now $S$ obtains exactly $x_e$ fractional token for each edge $e$ in $A$. If $A = \emptyset$, then $\chi(E(S)) = \sum_i \chi(E(R_i))$ which contradicts the linear independence of these sets of constraints in $\mathcal{L}$. Moreover, $x(A)$ is an integer and hence it is at least one, giving $S$ the one token it needs.

Since every edge is not integral, we have the extra fractional token of value $(1 - x_e)$ for every edge $e$ as unused tokens giving the contradiction.

□

## 4.3 Minimum Bounded-Degree Spanning Trees

We next turn to the study of the minimum bounded-degree spanning tree (MBDST) problem. In an instance of the MBDST problem we are given a graph $G = (V, E)$, edge cost given by $c : E \to \mathbb{R}$, a degree upper bound $B_v$ 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 originally due to Singh and Lau.

**Theorem 4.3.1** *There exists a polynomial time algorithm which given an instance of the MBDST problem returns a spanning tree $T$ such that $d_T(v) \leq B_v + 1$ and cost of the tree $T$ is at most the cost of any tree which satisfies the degree bounds.*

We prove Theorem 4.3.1 using the iterative relaxation technique. However, we first prove a weaker guarantee where the degree bound is violated by an additive factor of two, a result first obtained by Goemans and illustrates a simple extension of the leaf-finding iterative MST algorithm. Later in Section 4.4, we present the proof of Theorem 4.3.1 that extends the 1-edge-finding iterative MST algorithm.

### *4.3.1 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 vertex $v \in W$.

$$\begin{aligned}
\text{minimize} \quad & \sum_{e \in E} c_e \, x_e & & & & (4.5)\\
\text{subject to} \quad & x(E(V)) &= |V| - 1 & & & (4.6)\\
& x(E(S)) &\leq |S| - 1 & \quad \forall \emptyset \neq S \subset V & & (4.7)\\
& x(\delta(v)) &\leq B_v & \quad \forall \, v \in W & & (4.8)\\
& x_e &\geq 0 & \quad \forall \, e \in E & & (4.9)
\end{aligned}$$

Separation over the inequalities in the above linear program can be carried out in polynomial time and follows from Theorem 4.1.2. An alternative is to write a compact reformulation of the above linear program which has polynomially many variables and constraints (see the exercises).

### *4.3.2 Characterization of Extreme Point Solutions*

We first give a characterization of an extreme point solution to $LP_{mbdst}(G, \mathcal{B}, W)$. We remove all edges with $x_e = 0$ and focus only on the support of the extreme point solution

and the tight constraints from (4.6)-(4.8). Let $\mathcal{F} = \{S \subseteq V : x(E(S)) = |S| - 1\}$ be the set of tight constraints from (4.6)-(4.7). From an application of Rank Lemma 1.2.3 and the characterization of extreme point solutions to the spanning tree polyhedron (Lemma 4.1.5), we have the following characterization.

**Lemma 4.3.2** *Let $x$ be an extreme point solution to $LP_{mbdst}(G, \mathcal{B}, W)$ with $x_e > 0$ for each edge $e \in E$. Then there exists a set $T \subseteq W$ and a laminar family $\mathcal{L}$ such that*

(i) *$x(\delta(v)) = B_v$ for each $v \in T$ and $x(E(S)) = |S| - 1$ for each $S \in \mathcal{L}$.*
(ii) *The vectors in $\{\chi(E(S)) : S \in \mathcal{L}\} \cup \{\chi(\delta(v)) : v \in T\}$ are linearly independent.*
(iii) *$|\mathcal{L}| + |T| = |E|$.*

### *4.3.3 Leaf-finding Iterative Algorithm*

In this section, we give an iterative polynomial-time 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 4.5

**Iterative MBDST Leaf-finding Algorithm**

(i) Initialization $F \leftarrow \emptyset$.
(ii) While $|V(G)| \geq 2$ do
  (a) Find an optimal extreme point solution $x$ to $LP_{mbdst}(G, \mathcal{B}, W)$ and remove every edge $e$ with $x_e = 0$ from $G$. Let the support of $x$ be $E$.
  (b) If there exists a vertex $v \in V$ with at most one edge $e = uv$ incident on it, then update $F \leftarrow F \cup \{e\}$, $G \leftarrow G \setminus \{v\}$, $W \leftarrow W \setminus \{v\}$, and set $B_u \leftarrow B_u - 1$.
  (c) **(Relaxation)** If there exists a vertex $v \in W$ with $d_E(v) \leq 3$, then update $W \leftarrow W \setminus \{v\}$.
(iii) Return $F$.

Fig. 4.5. MBDST Leaf-finding Algorithm

### *4.3.4 Correctness and Performance Guarantee*

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

**Lemma 4.3.3** *Any extreme point solution $x$ to $LP_{mbdst}(G, \mathcal{B}, W)$ with $x_e > 0$ for each edge $e \in E$ must satisfy one of the following.*

*(a) There is a vertex $v \in V$ with $d_E(v) = 1$.*

*(b) There is a vertex $v \in W$ with $d_E(v) \leq 3$.*

*Proof* We use a global counting argument. Suppose for sake of contradiction that both (a) and (b) are not satisfied. Then every vertex has at least 2 edges incident on it and every vertex in $W$ has at least 4 edges incident on it. Therefore, $|E| = \frac{1}{2}\sum_{v \in V} d(v) \geq \frac{1}{2}(2(n - |W|) + 4|W|) = n + |W|$, where $n = |V|$.

On the other hand, by Lemma 4.3.2, there is a laminar family $\mathcal{L}$ and a set $T \subseteq W$ of vertices such that $|E| = |\mathcal{L}| + |T|$. As $\mathcal{L}$ is a laminar family which contains subsets of size at least two, from Proposition 4.1.7 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 performance guarantee of the algorithm.

**Theorem 4.3.4** *The leaf-finding iterative algorithm in Figure 4.5 returns a tree $T$ of optimal cost with $d_T(v) \leq B_v + 2$ 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 the optimality of the iterative spanning tree algorithm in Section 4.1 and we do not duplicate it here.

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

*Proof* The proof is by induction on the number of iterations of the algorithm. Clearly, $F = \emptyset$ 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 $v$. In the worst case, we may select all three edges in the solution. Hence,

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

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

## 4.4 An Additive One Approximation Algorithm

In this section, we give a very simple iterative algorithm which returns a tree of optimal cost and violates the degree bound within an additive error of one. This algorithm involves no rounding at all (not even picking an 1-edge) - it removes degree constraints one by one, and eventually reduces the problem to a minimum spanning tree problem. This can be thought of as an analogue of the 1-edge-finding iterative MST algorithm presented earlier. The algorithm is given in Figure 4.6.

**Iterative Relaxation MBDST Algorithm**

(i) While $W \neq \emptyset$ do

(a) Find an optimal extreme point solution $x$ to $LP_{mbdst}(G, \mathcal{B}, W)$ and remove every edge $e$ with $x_e = 0$ from $G$. Let the support of $x$ be $E$.

(b) **(Relaxation)** If there exists a vertex $v \in W$ with $d_E(v) \leq B_v + 1$, then update $W \leftarrow W \setminus \{v\}$.

(ii) Return $E$.

Fig. 4.6. Additive One MBDST Algorithm

### *4.4.1 Correctness and Performance Guarantee*

In the next lemma we prove that in each iteration, the algorithm can find some vertex for which the degree constraint can be removed. 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 4.1 to be integral. Hence, the algorithm returns a tree. Moreover, 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 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 $d_E(v) \leq B_v + 1$. Thus, in the worst case, if we include all the edges incident at $v$ in $T$, the degree bound of $v$ is violated by at most an additive one.

It remains to show that the iterative relaxation algorithm finds a degree constraint to remove at each step. From Lemma 4.3.2 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 $T = \emptyset$ then only the spanning tree inequalities define the solution $x$. Hence, $x$ must be integral. In the other case, we show that there must be a vertex in $W$ whose degree constraint can be removed.

**Lemma 4.4.1** *Let $x$ be an extreme point solution to $LP_{mbdst}(G, \mathcal{B}, W)$ with $x_e > 0$. Let $\mathcal{L}$ and $T \subseteq W$ correspond to the tight set constraints and the tight degree constraints defining $x$ as given by Lemma 4.3.2. If $T \neq \emptyset$ then there exists some vertex $v \in W$ with $d_E(v) \leq B_v + 1$.*

*Proof* We use the local fractional token argument as in the integrality proof of the 1-edge-finding iterative MST algorithm we presented earlier.

Suppose for the sake of contradiction, we have $T \neq \emptyset$ and $d_E(v) \geq B_v + 2$ for each $v \in W$. We now show a contradiction by a fractional 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 $|E| = |T| + |\mathcal{L}|$. The token redistribution is as follows. Each edge $e \in E$ gives as before $x_e$ fractional token

to the smallest set in $\mathcal{L}$ containing both endpoints of $e$, and $(1 - x_e)/2$ fractional token to each of its endpoints for the degree constraints.

We have already argued earlier that the $x_e$ assignment suffices to obtain one token for each member in the laminar family (see the third fractional token argument in the proof of Lemma 4.2.1).

Thus it suffices to show that each vertex with a tight degree constraint gets one token. Let $v \in T$ be such a vertex. Then $v$ receives $(1 - x_e)/2$ token for each edge incident at $v$ for a total token of value

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

where the first equality holds since $\sum_{e \in \delta(v)} x_e = B_v$ and the inequality holds since $d_E(v) \geq B_v + 2$ by Step (i)(b) of the algorithm.

To finish the proof, we argue that there is some extra token left for contradiction. 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 token which it does not need 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 the $(1 - x_e)/2 > 0$ token is extra. Note that $e \in \mathit{span}(\mathcal{L})$ for each $e$ with $x_e = 1$, since $e$ is a tight set of size two. We have

$$2\chi(E(V)) = \sum_{v \in V} \chi(\delta(v)) = \sum_{v \in T} \chi(\delta(v)) + \sum_{v \in V - T} \chi(\delta(v)) = \sum_{v \in T} \chi(\delta(v)) + \sum_{v \in V - T} \sum_{e \in \delta(v)} \chi(e).$$

We have argued that $V \in \mathcal{L}$ and $e \in \mathit{span}(\mathcal{L})$ for each edge $e \in \delta(v)$ for $v \in V - T$. Since $T \neq \emptyset$, this implies the linear independence of the tight constraints in $T$ and those in $\mathcal{L}$, giving us the contradiction. □

## Notes

Many variants of the greedy algorithm for finding minimum spanning trees have been obtained starting from Boruvka [17], Kruskal [93] and Prim [116] (Graham and Hell [65] have a useful survey of the history). Edmonds [41] gave the integral linear programming relaxation for minimum spanning tree problem that we presented.

There is a long line of work of successively improving the performance guarantees for the degree-bounded minimum-cost spanning tree problem. The algorithm with additive guarantee of one for the unweighted case was first given by Fürer and Raghavachari [57]. The additive algorithm with violation 2 (with both upper and lower degree bounds) was presented by Goemans [62]. The algorithm with additive violation of 1 was first presented by Singh and Lau [134], also for the case with upper and lower bounds on the degree. The fractional token proof which we used for the additive one proof was first presented by Bansal et al. [8].

## Exercises

4.1 **(Partition LP formulation for spanning trees.)** Consider the following partition LP for the minimum spanning tree problem. Let $\pi = \{V_1, \ldots, V_l\}$ be a partition of the vertex set $V$, and let $|\pi| = l$ denote the *size* of the partition. Define $\Delta(\pi)$ to be the set of edges with endpoints in different sets in the partition $\pi$. In any spanning tree, there are at least $|\pi| - 1$ edges in $\Delta(\pi)$ for a partition $\pi$ of $V$. Note that the undirected LP is a special case where only partitions of size two are considered. Show that the partition LP is equivalent to the subtour LP.

$$\begin{array}{lrcll} \text{minimize} & \displaystyle\sum_{e \in E} c_e \, x_e & & & \\ \text{subject to} & x(\Delta(\pi)) & \geq & |\pi| - 1 & \text{for all partitions } \pi \text{ of } V \\ & x(E(V)) & = & |V| - 1 & \\ & x_e & \geq & 0 & \forall \; e \in E \end{array}$$

4.2 Let $\tau = \{S \subseteq V : x(E(S)) = |S| - 1\}$ where $x$ is an extreme point solution to the spanning tree polyhedron. Solving the linear program using the ellipsoid method enables us to get a set family $\mathcal{F} \subseteq \tau$ such that constraints for sets in $\mathcal{F}$ are linearly independent and span all the tight constraints in $\tau$. But $\mathcal{F}$ need not be laminar. Give a polynomial time algorithm that, given $\mathcal{F}$, returns a laminar family $\mathcal{L} \subseteq \tau$ such that constraints for sets in $\mathcal{L}$ are independent and span all the tight constraints in $\tau$.

4.3 **(Spanning tree formulations.)** Another interesting formulation for the minimum spanning tree problem is the bidirected LP, which models the problem as finding a spanning arborescence (i.e. a directed spanning tree). 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 $V - r$. Given an undirected graph $G$, we construct a directed graph $D$ by having two opposite arcs $(u, v)$ and $(v, u)$ for each undirected edge $uv$ in $G$, and the cost of $(u, v)$ and $(v, u)$ in $D$ are the same as the cost of $uv$ in $G$. Then, pick an arbitrary vertex $r$ as the root vertex, and require that there is a directed path from $r$ to every vertex in $V - r$. Or equivalently, by Menger's theorem, require that there is at least one arc entering every set which does not contain the root vertex. The bidirected LP formulation for the minimum spanning tree problem is shown as follows.

$$\begin{array}{lrcll} \text{minimize} & \displaystyle\sum_{a \in A} c_a \, x_a & & & \\ \text{subject to} & x(\delta^{in}(S)) & \geq & 1 & \forall \; S \subseteq V - r \\ & x_a & \geq & 0 & \forall \; a \in A \end{array}$$

It is not clear a-priori whether the different spanning tree formulations are exact (i.e., give integer solutions at all extreme points) or not. (For example, consider

the bidirected LP which looks almost the same as the undirected LP.) Also, it is not clear whether they can be solved in polynomial time. The following theorem shows that these three formulations are equivalent.

**Theorem 4.4.2** *Suppose all edge weights are positive. Then the partition LP, the subtour LP, and the bidirected LP are equivalent. That is, any solution of one can be translated into a solution of the other with the same cost.*

Prove the above theorem and use it to provide separation oracles for these formulations hence proving the theorem below as well.

**Theorem 4.4.3** *There is a polynomial time separation oracle for the partition LP, the subtour LP, and the bidirected LP.*

4.4 Argue that in the iterative relaxation algorithm in Figure 4.6, one only needs to compute an optimal extreme point solution once initially, after that one can modify the current solution to obtain an extreme point solution for the next iteration in a simple way. (Hint: After we relax a degree constraint, we only need to find another extreme point solution with cost *no more* than the original solution, and the current solution is an "almost" extreme point solution for the next iteration.)

4.5 In an instance of the minimum bounded weighted-degree spanning tree we are given a graph $G = (V, E)$ and cost function $c : E \to \mathbb{R}^+$, a weight function $w : E \to \mathbb{R}^+$, a degree bound $B_v$ on each vertex $v$, and the task is to find a spanning tree $T$ with minimum cost and $\sum_{e \in \delta_T(v)} w(e) \leq B_v$ for all $v \in V$. Give a good bi-criteria approximation algorithm for the problem.

4.6 Can you generalize the result in the previous problem to get good bi-criteria approximation algorithm for the minimum bounded weighted-degree Steiner tree problem? Why or why not?

# 10
# Network Design

In this chapter we study the survivable network design problem. Given an undirected graph $G = (V, E)$ and a connectivity requirement $r_{uv}$ for each pair of vertices $u, v$, a *Steiner network* is a subgraph of $G$ in which there are at least $r_{uv}$ edge-disjoint paths between $u$ and $v$ for every pair of vertices $u, v$. The survivable network design problem is to find a Steiner network with minimum total cost. In the first part of this chapter, we will present the 2-approximation algorithm given by Jain [78] for this problem. We will present his original proof, which introduced the iterative rounding method to the design of approximation algorithms.

Interestingly, we will see a close connection of the survivable network design problem to the traveling salesman problem (TSP). Indeed the linear program, the characterization results, and presence of edges with large fractional value are identical for both problems. In the (symmetric) TSP we are given an undirected graph $G = (V, E)$ and cost function $c : E \to \mathbb{R}_+$ and the task is the find a minimum cost Hamiltonian cycle. In the second part of this chapter, we will present an alternate proof of Jain's result, which also proves a structural result about extreme point solutions to the traveling salesman problem.

In the final part of this chapter, we consider the minimum bounded degree Steiner network problem, where we are also given a degree upper bound $B_v$ for each vertex $v \in V$, and the task is to find a minimum cost Steiner network satisfying all the degree bounds. Using the idea of iterative relaxation, we show how to extend Jain's technique to this more general problem. We will first present a constant factor bicriteria approximation algorithm, and then show how to improve it to obtain additive approximation guarantee on the degree violation.

## 10.1 Survivable Network Design Problem

The survivable network design problem generalizes the minimum Steiner tree problem, the minimum Steiner forest problem, and the minimum $k$-edge-connected subgraph problem. Hence the results in this chapter also applies to these problems.

### *10.1.1 Linear Programming Relaxation*

To formulate the problem as a linear program, we represent the connectivity requirements by a skew supermodular function. As stated in Definition 2.3.10, a function $f : 2^V \rightarrow \mathbb{Z}$ is called skew supermodular if at least one of the following two conditions holds for any two subsets $S, T \subseteq V$.

$$\begin{aligned} f(S) + f(T) &\leq f(S \cup T) + f(S \cap T) \\ f(S) + f(T) &\leq f(S \backslash T) + f(T \backslash S) \end{aligned}$$

It can be verified that the function $f$ defined by $f(S) = max_{u \in S, v \notin S} \{r_{uv}\}$ for each subset $S \subseteq V$ is a skew supermodular function; see Exercise 2.9. Thus one can write the following linear programming relaxation for the survivable network design problem, denoted by $LP_{sndp}$, with the function $f$ being skew supermodular.

$$\begin{aligned} \text{minimize} \quad & \sum_{e \in E} c_e \, x_e & & & \\ \text{subject to} \quad & x(\delta(S)) & \geq & f(S) & \forall \, S \subseteq V \\ & 0 \leq x_e & \leq & 1 & \forall \, e \in E \end{aligned}$$

It is not known whether there is a polynomial time separation oracle for a general skew supermodular function $f$. This linear program for the minimum Steiner network problem, however, can be solved in polynomial time by using a minimum cut algorithm as a separation oracle.

### *10.1.2 Characterization of Extreme Point Solutions*

For a subset $S \subseteq V$, the corresponding constraint $x(\delta(S)) \geq f(S)$ defines a vector in $\mathbb{R}^{|E|}$: the vector has an 1 corresponding to each edge $e \in \delta(S)$, and a 0 otherwise. We call this vector the characteristic vector of $\delta(S)$, and denote it by $\chi(\delta(S))$. 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. By the strong submodularity of the cut function of undirected graphs (see Section 2.3.1), we have

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

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

for any two subsets $X$ and $Y$. When $f$ is skew supermodular, it follows from standard uncrossing arguments, as in spanning trees (Chapter 4) and maximum matchings (Chapter 9), that an extreme point solution to $LP_{sndp}$ is characterized by a laminar family of tight constraints. Lemma 10.1.1 below follows from these uncrossing arguments and the Rank Lemma.

**Lemma 10.1.1** *Let the requirement function $f$ of $LP_{sndp}$ be skew supermodular, and let $x$ be an extreme point solution to $LP_{sndp}$ with $0 < x_e < 1$ for every edge $e \in E$. Then, there exists a laminar family $\mathcal{L}$ such that:*

(i) $x(\delta(S)) = f(S)$ *for each* $S \in \mathcal{L}$.
(ii) *The vectors in* $\{\chi(\delta(S)) : S \in \mathcal{L}\}$ *are linearly independent.*
(iii) $|E| = |\mathcal{L}|$.

### *10.1.3 Iterative Algorithm*

The following is Jain's iterative rounding algorithm.

**Iterative Minimum Steiner Network Algorithm**

(i) Initialization $F \leftarrow \emptyset$, $f' \leftarrow f$;
(ii) While $f' \not\equiv 0$ do
  (a) Find an optimal extreme point solution $x$ to $LP_{sndp}$ with cut requirement $f'$ and remove every edge $e$ with $x_e = 0$.
  (b) If there exists an edge $e$ with $x_e \geq 1/2$, then add $e$ to $F$ and delete $e$ from the graph.
  (c) For every $S \subseteq V$: update $f'(S) \leftarrow \max\{f(S) - d_F(S), 0\}$.
(iii) Return $H = (V, F)$.

Fig. 10.1. Minimum Steiner Network Algorithm

### *10.1.4 Correctness and Performance Guarantee*

Jain developed a token counting argument and proved an important theorem about the extreme point solutions to $LP_{sndp}$.

**Theorem 10.1.2** *Suppose $f$ is an integral skew supermodular function and $x$ is an extreme point solution to $LP_{sndp}$. Then there exists an edge $e \in E$ with $x_e \geq \frac{1}{2}$.*

Assuming Theorem 10.1.2, the iterative algorithm will terminate, and it can be shown by an inductive argument that the returned solution is a 2-approximate solution (see Theorem 6.3.2 for a similar proof). We prove Theorem 10.1.2 by a counting argument, following the original proof of Jain, which will be useful later for the minimum bounded degree Steiner network problem.

Suppose for a contradiction that every edge $e$ has $0 < x_e < \frac{1}{2}$. By Lemma 10.1.1, there is a laminar family $\mathcal{L}$ of tight constraints that defines $x$. We assign two tokens to each edge, one to each endpoint, for a total of $2|E|$ tokens. Then we will redistribute the

tokens so that each member in $\mathcal{L}$ receives at least 2 tokens and there are some tokens left. This would imply that $|E| > |L|$, contradicting Lemma 10.1.1.

Let $S$ be a subset of vertices. Define its co-requirement $coreq(S) := \sum_{e \in \delta(S)} (\frac{1}{2} - x_e)$. We call a set $S \in \mathcal{L}$ *special* if $coreq(S) = \frac{1}{2}$. Given a laminar family, it can be represented as a forest of trees if its sets are ordered by inclusion. Recall that in this forest, a set $R \in \mathcal{L}$ is a child of $S \in \mathcal{L}$ if $S$ is the smallest set in $\mathcal{L}$ that contains $R$. We say an endpoint $v$ is *owned* by $S$ if $S$ is the smallest set in $\mathcal{L}$ that contains $v$. The following lemma is useful to establish that a certain set is special. A proof can be found in Vazirani's book [140], and it is omitted here.

**Lemma 10.1.3** *Suppose $S$ has $\alpha$ children and owns $\beta$ endpoints where $\alpha + \beta = 3$. If all children of $S$ are special, then $S$ is special.*

The redistribution of tokens is by an inductive argument based on the following lemma, which would yield the contradiction that $|E| > |L|$.

**Lemma 10.1.4** *For any rooted subtree of the forest $\mathcal{L}$ with root $S$, the tokens assigned to vertices in $S$ can be redistributed such that every member in the subtree gets at least two tokens, and the root $S$ gets at least three tokens. Furthermore, $S$ gets exactly three tokens only if $S$ is special; otherwise $S$ gets at least four tokens.*

*Proof* The proof is by induction. For the base case, consider a leaf node $S$ in the laminar family. Since $f(S) \geq 1$ and $x_e < \frac{1}{2}$ for all $e$, this implies that $d(S) \geq 3$ and thus $S$ can collect three tokens. Furthermore, $S$ collects exactly three tokens only if $f(S) = 1$ and $d(S) = 3$, in which case $S$ is special. This verifies the base case.

For the induction step, consider a non-leaf node $S$. For a child $R$ of $S$, we say $R$ has one excess token if $R$ has three tokens, and $R$ has at least two excess tokens if $R$ has at least four tokens. By the induction hypothesis, each child has at least one excess token, and has exactly one excess token only if it is special. We will divide the cases by the number of children of $S$.

(i) $S$ has at least 4 children: Then $S$ can collect 4 tokens by taking one excess token from each child.

(ii) $S$ has 3 children: If any child of $S$ has two excess tokens or if $S$ owns any endpoint, then $S$ can collect 4 tokens. Otherwise, all three children of $S$ are special and $S$ does not own any enpdoint, but then $S$ is special by Lemma 10.1.3, and so 3 tokens are enough for the induction hypothesis.

(iii) $S$ has 2 children: If both children have two excess tokens, then $S$ can collect 4 tokens. So assume that one child of $S$ is special, but then it can be shown that $S$ must own at least one endpoint (see the exercises or Lemma 23.20 of [140]). If both children of $S$ are special, then $S$ is special by Lemma 10.1.3, and so three tokens are enough for the induction hypothesis. Otherwise $S$ can collect 4 tokens.

(iv) $S$ has one child $R$: Since $\chi(\delta(S))$ and $\chi(\delta(R))$ are linearly independent, $S$ must owns at least one endpoint. As both $f(S)$ and $f(R)$ are integers and there is no edge of integral value, this actually implies that $S$ cannot own exactly one endpoint, and thus $S$ owns at least two endpoints. If $R$ is not special, then $S$ can collect 4 tokens; otherwise, $S$ is special by Lemma 10.1.3 and so 3 tokens are enough for the induction hypothesis.

□

The extra tokens in the roots of the laminar family give us the desired contradiction. This proves Theorem 10.1.2, and thus the 2-approximation algorithm for the survivable network design problem follows.

## 10.2 Connection to the Traveling Salesman Problem

In this section we formulate a generalization of the survivable network design problem and the traveling salesman problem. Then we present a proof that any extreme point solution of this generalization has an edge $e$ with $x_e \geq \frac{1}{2}$, and in some cases, $x_e = 1$. This will generalize Jain's result and a result of Boyd and Pulleyblank on the traveling salesman problem. The new proof uses the fractional token technique, and does not need the notion of special sets used in the previous section to prove Jain's result.

### *10.2.1 Linear Programming Relaxation*

The following is a linear programming relaxation which models both the survivable network design problem and the traveling salesman problem, denoted by $LP_f$, where $f$ is a skew supermodular function. It is similar to the linear programming relaxation for the minimum bounded degree Steiner network problem in the next section.

$$
\begin{array}{lrcll}
\text{minimize} & \displaystyle\sum_{e \in E} c_e \, x_e & & & \\
\text{subject to} & x(\delta(S)) & \geq & f(S) & \forall \, S \subseteq V \\
& x(\delta(v)) & = & f(v) & \forall \, v \in W \\
& 0 \leq x_e & \leq & 1 & \forall \, e \in E
\end{array}
$$

For the survivable network design problem we set $f(S) = max_{u \in S, v \notin S} \, r_{uv}$ for each subset $S \subseteq V$ and set $W = \emptyset$. For the traveling salesman problem we set $f(S) = 2$ for each $S \subset V$, $f(V) = 0$ and $W = V$. Note that the same characterization of extreme point solutions as in Lemma 10.1.1 holds for $LP_f$.

### *10.2.2 Existence of Edges with Large Fractional Value*

Boyd and Pulleyblank [18] proved that there is an edge $e$ with $x_e = 1$ in any extreme point solution to the traveling salesman problem. The following theorem provides a common generalization to their result and Jain's result (Theorem 10.1.2).

**Theorem 10.2.1** *Let $f$ be an integral skew supermodular function, and $x$ be an extreme point solution to $LP_f$ with $x_e > 0$ for all $e$. Then there exists an $e \in E$ with $x_e \geq \frac{1}{2}$. Moreover, if $f(S)$ is an even integer for each subset $S \subseteq V$ then there exists an edge $e$ with $x_e = 1$.*

*Proof* **[First part]:** We first prove that $x_e \geq \frac{1}{2}$ for some edge $e \in E$ in any extreme point solution $x$ to $LP_f$. Suppose for a contradiction that $0 < x_e < \frac{1}{2}$ for each $e \in E$. Then we will show that $|E| > |\mathcal{L}|$, contradicting Lemma 10.1.1. The proof is by a fractional token counting argument. We give one token to each edge in $E$, and then we will reassign the tokens such that we can collect one token for each member in $\mathcal{L}$ and still have extra tokens left, giving us the contradiction that $|E| > |\mathcal{L}|$. Each edge $e = uv$ is given one token which is reassigned as follows.

(i) **(Rule 1)** Let $S \in \mathcal{L}$ be the smallest set containing $u$ and $R \in \mathcal{L}$ be the smallest set containing $v$. Then $e$ gives $x_e$ tokens each to $S$ and $R$.

(ii) **(Rule 2)** Let $T$ be the smallest set containing both $u$ and $v$. Then $e$ gives $1 - 2x_e$ tokens to $T$.

We now show that each set $S$ in $\mathcal{L}$ receives at least one token. Let $S$ be any set with children $R_1, \ldots, R_k$ where $k \geq 0$ (if $S$ does not have any children then $k = 0$). We have the following equalities.

$$\begin{aligned} x(\delta(S)) &= f(S) \\ x(\delta(R_i)) &= f(R_i) \ \ \forall\, 1 \leq i \leq k \end{aligned}$$

Subtracting we obtain,

$$x(\delta(S)) - \sum_{i=1}^{k} x(\delta(R_i)) = f(S) - \sum_{i=1}^{k} f(R_i). \tag{10.1}$$

We divide the edges involved in the left hand side of (10.1) into three types, where

$$\begin{aligned} A &= \{e : |e \cap (\cup_i R_i)| = 0, |e \cap S| = 1\} \\ B &= \{e : |e \cap (\cup_i R_i)| = 1, |e \cap S| = 2\} \\ C &= \{e : |e \cap (\cup_i R_i)| = 2, |e \cap S| = 2\}. \end{aligned}$$

Then (10.1) can be rewritten as:

$$x(A) - x(B) - 2x(C) = f(S) - \sum_{i=1}^{k} f(R_i). \tag{10.2}$$

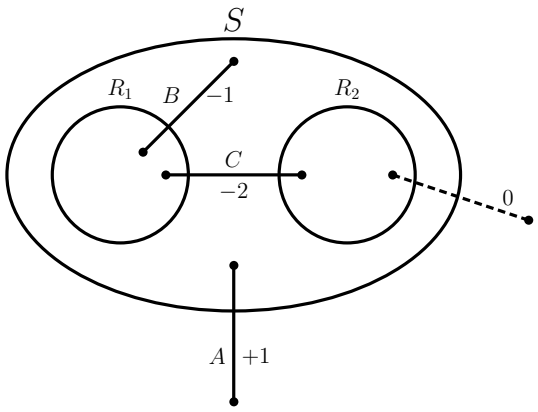


Fig. 10.2. Example for expression $x(\delta(S)) - \sum_{i=1}^{k} x(\delta(R_i))$ with $k = 2$ children. The dashed edges cancel out in the expression. Edge-sets $A, B, C$ shown with their respective coefficients.

Observe that $A \cup B \cup C \neq \emptyset$; otherwise the characteristic vectors $\chi(\delta(S))$, $\chi(\delta(R_1))$, $\ldots, \chi(\delta(R_k))$ are linearly dependent. For each edge $e \in A$, $S$ receives $x_e$ tokens from $e$ by Rule 1. For each edge $e \in B$, $S$ receives $1 - x_e$ tokens from $e$ by Rule 1 and Rule 2. For each edge $e \in C$, $S$ receives $1 - 2x_e$ tokens from $e$ by Rule 2. Hence, the total tokens received by $S$ is exactly

$$\begin{aligned} & \sum_{e \in A} x_e + \sum_{e \in B} (1 - x_e) + \sum_{e \in C} (1 - 2x_e) \\ = \; & x(A) + |B| - x(B) + |C| - 2x(C) \\ = \; & |B| + |C| + f(S) - \sum_{i=1}^{k} f(R_i), \end{aligned}$$

where the last equality follows from (10.2). Since $f$ is integral and the above expression is non-zero (since $A \cup B \cup C \neq \emptyset$), the right hand side is at least one, and thus every set $S \in \mathcal{L}$ receives at least one token in the reassignment.

It remains to show that there are some unassigned tokens, which would imply the contradiction that $|E| > |\mathcal{L}|$. Let $R$ be any maximal set in $\mathcal{L}$. Consider any edge $e \in \delta(R)$. The fraction of the token by Rule 2 for edge $e$ is unassigned, as there is no set $T \in \mathcal{L}$ with $|T \cap e| = 2$, gives us the desired contradiction. This proves the first part of the theorem.

**[Second part]:** The proof of the second part is almost identical to the proof of the first part, except that we use scaled token assignment rules. Each edge $e = uv$ is given one token which it reassigns as follows.

(i) **(Rule 1')** Let $S$ be the smallest set containing $u$ and $R$ be the smallest set containing $v$ in $\mathcal{L}$. Then $e$ gives $x_e/2$ tokens each to $S$ and $R$.

(ii) **(Rule 2')** Let $T$ be the smallest set containing both $u$ and $v$. Then $e$ gives $1 - x_e$ tokens to $T$.

We now show that each set in $\mathcal{L}$ receives at least one token. Let $S$ be any set with children $R_1, \ldots, R_k$ where $k \geq 0$. We have the following equalities.

$$\begin{aligned} x(\delta(S)) &= f(S) \\ x(\delta(R_i)) &= f(R) \ \ \forall\, 1 \leq i \leq k \end{aligned}$$

Dividing by two and subtracting we obtain,

$$\frac{x(\delta(S)) - \sum_i x(\delta(R_i))}{2} = \frac{f(S) - \sum_i f(R_i)}{2}$$

The edges involved in the left hand side are divided into types $A, B, C$ exactly as in the first part. Then the above equation becomes

$$\frac{x(A) - x(B)}{2} - x(C) = \frac{f(S) - \sum_i f(R_i)}{2}.$$

Observe that $A \cup B \cup C \neq \emptyset$; otherwise there is a linear dependence among the constraints for $S$ and its children. Also, $S$ receives $\frac{x_e}{2}$ tokens for each edge $e \in A$ by Rule 1', $1 - \frac{x_e}{2}$ tokens for each edge $e \in B$ by Rule 1' and Rule 2', and $1 - x_e$ tokens for each edge $e \in C$ by Rule 2'. Hence, the total tokens received by $S$ is exactly

$$\begin{aligned} & \sum_{e \in A} \frac{x_e}{2} + \sum_{e \in B} (1 - \frac{x_e}{2}) + \sum_{e \in C} (1 - x_e) \\ = \ & \frac{x(A)}{2} + |B| - \frac{x(B)}{2} + |C| - x(C) \\ = \ & |B| + |C| + \frac{f(S) - \sum_i f(R_i)}{2}. \end{aligned}$$

Since $A \cup B \cup C \neq \emptyset$ and $f$ is an even-valued function, this is a positive integer . Thus every set $S \in \mathcal{L}$ receives at least one token in the reassignment.

Now, we show that there are some unassigned tokens showing the strict inequality $|\mathcal{L}| < |E|$. Let $R$ be any maximal set $\mathcal{L}$. Then, consider any edge $e \in \delta(R)$. Then, token by Rule 2' for edge $e$ is unassigned as there is no set $T \in \mathcal{L}$ with $|T \cap e| = 2$. This gives us the desired contradiction. □

### 10.3 Minimum Bounded Degree Steiner Networks

In general, the minimum bounded degree Steiner network problem does not admit any polynomial factor approximation algorithm, since the minimum cost Hamiltonian cycle problem is a special case. Instead, we will present *bicriteria* approximation algorithms, that both violate the degree bounds as well as delivers a suboptimal cost solution. In this section, we will first present a $(2, 2B_v + 3)$-approximation algorithm for the minimum bounded degree Steiner network problem, where $B_v$ denotes the degree upper bound on node $v$. By this, we mean that the algorithm outputs a Steiner network whose cost is at most twice that of the objective value of the linear programming relaxation, and whose

degree at any node $v$ is at most $2B_v + 3$. In the next section, we will present an algorithm with only an additive violation on the degree bounds.

### *10.3.1 Linear Programming Relaxation*

The linear programming formulation for the minimum bounded degree Steiner network problem, denoted by $LP_{bdsn}$, is a straightforward generalization of $LP_{sndp}$ for the survivable network design problem, with $f(S) = max_{u \in S, v \notin S} \{r_{uv}\}$ for each subset $S \subseteq V$. Notice that the degree constraints are only present on a subset $W \subseteq V$ of vertices.

$$
\begin{array}{lrcll}
\text{minimize} & \displaystyle\sum_{e \in E} c_e \, x_e & & & \\
\text{subject to} & x(\delta(S)) & \geq & f(S), & \forall \, S \subseteq V \\
& x(\delta(v)) & \leq & B_v, & \forall \, v \in W \\
0 \leq & x_e & \leq & 1 & \forall \, e \in E
\end{array}
$$

### *10.3.2 Characterization of Extreme Point Solutions*

As the degree constraints are defined only on single vertices, the same uncrossing technique (as in Lemma 10.1.1) can be applied to show that an optimal extreme point solution is characterized by a laminar family of tight constraints.

**Lemma 10.3.1** *Let the requirement function of $LP_{bdsn}$ be skew supermodular, and let $x$ be an extreme point solution to $LP_{bdsn}$ with $0 < x_e < 1$ for every edge $e \in E$. Then, there exists a laminar family $\mathcal{L}$ of tight sets such that $\mathcal{L}$ partitions into a set of singletons $\mathcal{L}'$ for the degree constraints, and the remaining sets $\mathcal{L}'' = \mathcal{L} - \mathcal{L}'$ for the connectivity constraints, such that:*

(i) $x(\delta(v)) = B_v > 0$ *for all* $\{v\} \in \mathcal{L}'$ *and* $x(\delta(S)) = f(S) \geq 1$ *for all* $S \in \mathcal{L}''$.

(ii) *The vectors in* $\{\chi(\delta(S)) : S \in \mathcal{L}\}$ *are linearly independent.*

(iii) $|E| = |\mathcal{L}|$.

### *10.3.3 Iterative Algorithm*

The iterative algorithm is similar to that for the minimum Steiner network problem, with a new relaxation step where we remove degree constraints of vertices with low degree.

**Iterative Minimum Bounded Degree Steiner Network Algorithm**

(i) Initialization $F \leftarrow \emptyset$, $f' \leftarrow f$.
(ii) While $f' \not\equiv 0$ do
  (a) Find an optimal extreme point solution $x$ with cut requirement $f'$ and remove every edge $e$ with $x_e = 0$.
  (b) **(Relaxation):** If there exists a vertex $v \in W$ with degree at most 4, then remove $v$ from $W$.
  (c) **(Rounding):** If there exists an edge $e = (u, v)$ with $x_e \geq 1/2$, then add $e$ to $F$ and delete $e$ from the graph and decrease $B_u$ and $B_v$ by 1/2.
  (d) For every $S \subseteq V$: update $f'(S) \leftarrow \max\{f(S) - d_F(S), 0\}$.
(iii) Return $H = (V, F)$.

Fig. 10.3. Minimum Bounded Degree Steiner Network Algorithm

### *10.3.4 Correctness and Performance Guarantee*

The performance guarantee follows by an inductive argument as in Lemma 6.3.2, assuming the algorithm terminates. We will prove the following lemma showing that the algorithm will terminate.

**Lemma 10.3.2** *Let $x$ be an extreme point solution to $LP_{bdsn}$ with $x_e > 0$ for every edge $e$, and $W$ be the set of vertices with degree constraints. Then either one of the following holds:*

(i) *There exists an edge $e$ with $x_e \geq \frac{1}{2}$.*
(ii) *There exists a vertex $v \in W$ with degree $d(v) \leq 4$.*

In the following we use a token counting argument to prove Lemma 10.3.2. The counting argument is similar to that of Theorem 10.1.2. Each edge is assigned two tokens, for a total of $2|E|$ tokens. For each edge $e$, one token is assigned to each endpoint. We shall show that if none of the steps in the algorithm can be applied, then each set in $\mathcal{L}''$ and each degree constraint in $\mathcal{L}'$ can collect two tokens, and there are some tokens left. This would imply $|E| > |\mathcal{L}|$, contradicting that $x$ is an extreme point solution.

We prove the same statement as in Lemma 10.1.4 to get the contradiction. The counting argument is similar and we only highlight the differences here. By Step 2(b), we may assume that $d(v) \geq 5$ for each $v \in W$, and hence each degree constraint has three extra tokens. The case when the node $S \in \mathcal{L}$ does not own a vertex in $W$ is exactly the same as in Lemma 10.1.4. Henceforth we consider a node $S \in \mathcal{L}$ that owns a vertex $v \in W$. If $S$ has another child $R$, then $S$ can collect three tokens from $v$ and at least one token from $R$, as desired. Otherwise, by linear independence of $\chi(\delta(S))$ and $\chi(\delta(v))$, $S$ owns at least one endpoint and thus $S$ can collect three tokens from $v$ and at least one more token from the endpoints it owns as required. This completes the proof of Lemma 10.3.2.

## 10.4 An Additive Approximation Algorithm

In this section we show how to achieve additive guarantee on the degree bounds that depend only on the maximum connectivity requirement, denoted by $r_{max} = \max_{u,v}\{r_{uv}\}$.

**Theorem 10.4.1** *There is a polynomial time* $(2, B_v + 6r_{\max} + 3)$*-approximation algorithm for the minimum bounded degree Steiner network problem, which returns a solution with cost at most twice the optimum while the degree of each vertex* $v$ *is at most* $B_v + 6r_{\max} + 3$.

For minimum bounded degree Steiner trees, since the maximum connectivity requirement is one, this algorithm will output a solution that violates the degree bounds only by an additive constant, and of cost within twice the optimal.

### *10.4.1 Iterative Algorithm*

The iterative algorithm uses the same linear programming relaxation $LP_{bdsn}$ as in the previous section. The difference is that we only pick an edge $e$ with $x_e \geq \frac{1}{2}$ when both endpoints have "low" degrees.

**Additive Approximation for Minimum Bounded Degree Steiner Network**

(i) Initialization $F \leftarrow \emptyset$, $f' \leftarrow f$.
(ii) While $f' \not\equiv 0$ do
  (a) Find an optimal extreme point solution $x$ to $LP_{bdsn}$ satisfying $f'$ and remove every edge $e$ with $x_e = 0$. Set $W_h = \{v \in W \mid \sum_{e \in \delta(v)} x_e \geq 6r_{max}\}$ and $B_v = \sum_{e \in \delta(v)} x_e$ for $v \in W$.
  (b) If there exists an edge $e = (u, v)$ with $x_e = 1$, then add $e$ to $F$ and remove $e$ from $G$ and decrease $B_u$ and $B_v$ by 1.
  (c) **(Relaxation):** If there exists a vertex $v \in W$ with degree at most 4, then remove $v$ from $W$.
  (d) **(Rounding):** If there exists an edge $e = (u, v)$ with $x_e \geq \frac{1}{2}$ and $u, v \notin W_h$, then add $e$ to $F$ and remove $e$ from $G$ and decrease $B_u$ and $B_v$ by $x_e$.
  (e) For every $S \subseteq V$: $f'(S) \leftarrow \max\{f(S) - d_F(S), 0\}$.
(iii) Return $H = (V, F)$.

Fig. 10.4. Additive Approximation for Minimum Bounded Degree Steiner Network

### *10.4.2 Correctness and Performance Guarantee*

First we show that the degree of any vertex $v$ in the returned solution $H$ is at most $B_v + 6r_{max} + 3$, assuming that the algorithm terminates. We define the set $W_h$ of vertices

(defined in Step (ii)(a) of the algorithm)with fractional degree at least $6r_{\max}$ as *high* degree vertices. Observe that the fractional degree of each vertex is non-increasing during the algorithm, since we reset the degree upper bound in Step (ii)(a) in each iteration. Consider an edge $e$ with $v$ as an endpoint. When $v \in W_h$, $e$ is picked only if $x_e = 1$ in Step (ii)(b) of the algorithm. Hence, while $v \in W_h$, at most $B_v - 6r_{max}$ edges incident at $v$ are added to $H$. While $v \in W \setminus W_h$, $e$ is picked only if $x_e \geq \frac{1}{2}$ in Step (ii)(d) of the algorithm. Hence, while $v \in W \setminus W_h$, strictly less than $12r_{max}$ edges incident at $v$ are added to $H$. Finally, by Step (ii)(c) of the algorithm, a degree constraint is removed only if $v$ is incident to at most four edges, where possibly all of them are added to $H$. Therefore, the degree of $v$ in $H$ is strictly less than $(B_v - 6f_{max}) + 12f_{max} + 4 = B_v + 6f_{max} + 4$. As $B_v$ is an integer, the degree of $v$ in $H$ is at most $B_v + 6f_{max} + 3$. Moreover, since we always included edges with value at least half in $F$ in each iteration, by induction, the cost of the final solution is at most twice the optimal objective value of $LP_{bdsn}$. This proves Theorem 10.4.1, assuming that the algorithm terminates.

The following lemma proves that the algorithm will always terminate. With the same characterization as in Lemma 10.3.1, we use a more careful counting argument to prove stronger properties of the extreme point solutions to $LP_{bdsn}$ than those in Lemma 10.3.2.

**Lemma 10.4.2** *Let $x$ be an extreme point solution to $LP_{bdsn}$ with $x_e > 0$ for all $e$, and $W$ be the set of vertices with degree constraints, and $W_h = \{v \in W \mid \sum_{e \in \delta(v)} x_e \geq 6r_{max}\}$. Then at least one of the following holds.*

(i) *There exists an edge $e$ with $x_e = 1$.*
(ii) *There exists an edge $e = \{u, v\}$ with $x_e \geq 1/2$ and $u, v \notin W_h$.*
(iii) *There exists a vertex $v \in W$ with degree at most 4.*

We will prove Lemma 10.4.2 by a token counting argument as in Lemma 10.3.2. Suppose for contradiction that none of the conditions in Lemma 10.4.2 holds. Then each edge $e$ has $x_e < 1$, and each edge $e$ with $\frac{1}{2} \leq x_e < 1$ (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 to each edge 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 $\mathcal{L}'$ gets at least two tokens, and there are still some excess tokens left. This will imply that $|E| > |\mathcal{L}|$, contradicting Lemma 10.3.1.

The main difference from Lemma 10.3.2 is the existence of heavy edges (with an endpoint in $W_h$) which our algorithm is not allowed to pick. Since there exist 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 Lemma 10.1.4. To get around this, we use a different token assignment scheme and revise the definition of co-requirement, so that a similar induction hypothesis would work.

**Token assignment scheme:** If $e = uv$ is a heavy edge with $u \in W_h$ 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 revise the definition of co-requirement for the presence of heavy edges:

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

It is useful to note that this definition reduces to the original definition of co-requirement, if every edge $e$ with $x_e \geq \frac{1}{2}$ is thought of as two parallel edges, each aiming to achieve a value of $\frac{1}{2}$ and each has fractional value $\frac{x_e}{2}$: summing $\frac{1}{2} - \frac{x_e}{2}$ over both edges gives $1 - x_e$. We say a set is *special* if $coreq(S) = \frac{1}{2}$ as in the proof of Jain's theorem.

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. However, by exploiting the fact that $f(S) \leq r_{max}$, we will show that vertices in $W_h$ can *get back* enough tokens. We prove the following lemma which shows that the tokens can be reassigned in a similar way as in Lemma 10.1.4.

**Lemma 10.4.3** *For any subtree of $\mathcal{L}$ rooted at $S$, we can reassign tokens such that each 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 special; otherwise it gets at least four tokens.*

The following is a key claim showing if $S$ owns a vertex in $W_h$, then there are enough tokens for $S$ and the vertices in $W_h$ that it owns.

**Claim 10.4.4** *Suppose $S$ owns $w \geq 1$ 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 $W_h \cap S$ is at least $2w + 4$.*

*Proof* Suppose $S$ has $c$ children. As each child has at least one excess token by the induction hypothesis, if $c \geq 6w$ then we have $6w$ tokens which is at least $2w + 4$. Henceforth we assume that $c < 6w$. Let $B := \sum_v x(\delta(v))$, where the sum is over all vertices $v \in W_h$ owned by $S$. Note that $B \geq 6wr_{max}$ by the definition of $W_h$. For a child $R$ of $S$, as $x(\delta(R)) = f(R) \leq r_{max}$, at most $r_{max}$ units of $B$ are contributed by the edges in $\delta(R)$. Similarly, at most $r_{max}$ units of $B$ are contributed by the edges in $\delta(S)$. Hence, at least $r_{max}(6w - c - 1)$ units of $B$ are from the edges with both endpoints owned by $S$. Since there is no edge $e$ with $x_e = 1$, there are at least $r_{max}(6w - c - 1) + 1$ such edges. Let $e = uv$ 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 $S$. Hence, the number of tokens owned by $S$ plus the number of tokens left with vertices in $W_h$ owned by $S$ is at least $r_{max}(6w - c - 1) + 1$. Furthermore, $S$ can also collect one excess token from each child.

So, the total number of tokens that $S$ can collect is at least $r_{max}(6w-c-1)+c+1$, which is a decreasing function of $c$. As $c < 6w$, the number of tokens is minimized at $c = 6w-1$, which is at least $2w+4$, proving the claim. □

We now proceed by induction on the height of the subtree to prove Lemma 10.4.3.

**Base Case:** $S$ is a leaf node. Claim 10.4.4 implies that $S$ can collect enough tokens if it owns vertices in $W_h$. Hence assume $S \cap W_h = \emptyset$. First consider the case when $S \cap W \neq \emptyset$. Any vertex $v \in W \setminus W_h$ has at least five tokens, and thus has three excess tokens. If $S$ owns two such vertices or $S$ owns another endpoint, then $S$ gets at least four tokens as required. Otherwise, we have $\chi(\delta(v)) = \chi(\delta(S))$, contradicting Lemma 10.3.1.

Henceforth we consider the case when $S \cap W = \emptyset$. Then $S$ can get at least $|\delta(S)| = d(S)$ tokens from the vertices owned by $S$. Note that $d(S) \geq 2$, as $x(\delta(S))$ is an integer and there is no edge $e$ with $x_e = 1$. If $d(S) \geq 4$, then $S$ gets at least four tokens. If $d(S) = 3$ and $d(S)$ contains a heavy edge, then $S$ can get four tokens from the vertices it owns. If it does not contain a heavy edge, then $S$ receives three tokens and $coreq(S) = \frac{1}{2}$, for which three tokens are enough. If $d(S) = 2$, then at least one edge is a heavy edge. If both edges are heavy then $S$ can get four tokens; otherwise if only one edge is heavy then $coreq(S) = \frac{1}{2}$ and so three tokens are enough.

**Induction Step:** If $S$ owns any vertex in $W_h$, then we are done by Claim 10.4.4. Thus we can assume that $S$ does not own any vertex in $W_h$. By the induction hypothesis, any child of $S$ has at least one excess token. So, if $S$ owns a vertex in $W \setminus W_h$, then $S$ can collect at least four tokens. Hence, we can assume that $S$ does not own any vertex in $W$, and the proof of these cases are almost identical to that in Lemma 10.1.4 with the definition of co-requirement revised; the details are omitted.

### *10.4.3 Steiner Forests*

In the special cases of Steiner trees and Steiner forests, the connectivity requirement function is a $\{0,1\}$-function and hence $r_{\max} = 1$. Theorem 10.4.1 thus implies that there is a $(2, B_v + 9)$-approximation algorithm for this problem, but it is possible to obtain a better bound.

**Theorem 10.4.5** *There is a polynomial time $(2, B_v + 3)$-approximation algorithm for the minimum bounded degree Steiner network problem when $r_{\max} = 1$.*

The iterative algorithm is similar where the improvement comes from the following fact [96, 133]: there is a heavy edge between two vertices without degree constraints. The proof is by a more involved counting argument that uses a different induction hypothesis.

## Notes

The algorithm for the survivable network design problem is due to Jain [78], who introduced the iterative rounding method in the design of approximation algorithms. The existence of an 1-edge in the traveling salesman problem is due to Boyd and Pulleyblank [18], and the proof presented in this chapter is from the work of Nagarajan, Ravi and Singh [110]. The algorithm for the minimum bounded degree Steiner network problem is by Lau, Naor, Salavatipour and Singh [95], who first used the iterative relaxation idea in degree-bounded network design problems. The subsequent work with additive violation on the degree bounds is by Lau and Singh [96].

## Exercises

10.1 Prove Lemma 10.1.3.

10.2 Show the following statement in the proof of Lemma 10.1.4: If set $S$ has two children, one of which has a co-requirement of 1/2, then it must own at least one endpoint.

10.3 Give a 2-approximation algorithm for the traveling salesman problem, using the fact there is always an edge $e$ with $x_e = 1$ in the linear programming relaxation $LP_f$.

10.4 Consider the subtour elimination LP for asymmetric TSP [71]. Find the largest value $\rho$ such that there is always some arc variable with $x_a \geq \rho$.

10.5 Design a bicriteria approximation algorithm for the minimum bounded degree strongly $k$-arc-connected subgraph problem in directed graphs.

(a) Write a "cut-cover" linear programming relaxation for the problem. Show that the connectivity requirement function for this problem is a crossing supermodular function.

(b) Use the uncrossing technique to show that an extreme point solution is defined by a cross-free family of tight constraints.

(c) Prove that there is an arc with value at least $\frac{1}{3}$ in any extreme point solution when there are no degree constraints in the problem.

(d) Apply the iterative relaxation step to obtain a bicriteria approximation algorithm for the problem.

10.6 Let $P$ be the Peterson graph, and let $w_1, w_2, w_3$ be the neighbors of a vertex $v$ in $P$. Take three copies of $P - v$ and three new vertices $v_1, v_2, v_3$, and attach them as follows: for $1 \leq j \leq 3$, add edges from $v_j$ to each $w_j$ in the three copies of $P - v$. Prove that the resulting graph is a 3-regular 3-edge-connected graph with no Hamiltonian cycle. By setting $B_v = 1$ for each vertex and connectivity requirement $r_{uv} = 1$ for each pair of vertices, show that this is an integrality gap example in which there is a feasible solution to $LP_{bdsn}$ but there is no Steiner network with degree at most $2B_v$ (or $B_v + 1$) for each vertex.

10.7 Show that the following example gives an integrality gap of at least an additive $n/4$ for the degree violation for $LP_{bdsn}$. The input is a complete bipartite graph $B = (X, Y, E)$ where $X = \{x_1, x_2\}$ and $Y = \{y_1, \ldots, y_n\}$. 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 $\frac{n}{2}$, and 0 otherwise.

10.8 Can you use the fractional token technique in Section 10.2 to obtain the results for the minimum bounded degree Steiner network problem presented in this chapter?