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.

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 that 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:

$$\begin{aligned} &\text{minimize} && \sum_{u,v} w_{uv} x_{uv} \\ &\text{subject to} && \sum_{v:\{u,v\}\in E} x_{uv} = 1 \quad \forall u \in V_1 \end{aligned}$$

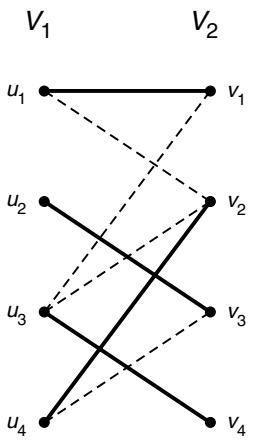

Figure 1.1 The solid edges form a perfect matching in the bipartite graph.

$$\sum_{u:\{u,v\}\in E} x_{uv} = 1 \qquad \forall v \in V_2$$

$$x_{uv} \geq 0 \qquad \forall \{u,v\} \in E$$

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 [71] 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 formulated earlier 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 preceding linear program (LP) 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 [121]. 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, 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 these iterations till all variables have been fixed to either 0 or 1. Given the preceding iterative algorithm, there are two claims that need to be proven. First, the algorithm works correctly (i.e., it can always find a variable with value 0 or 1) in each iteration, and, second, the selected matching 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 programs to show that the preceding iterative algorithm works correctly.

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 nonzero vector $y \in \mathbb{R}^n$ such that $x + y, x - y \in P$.*

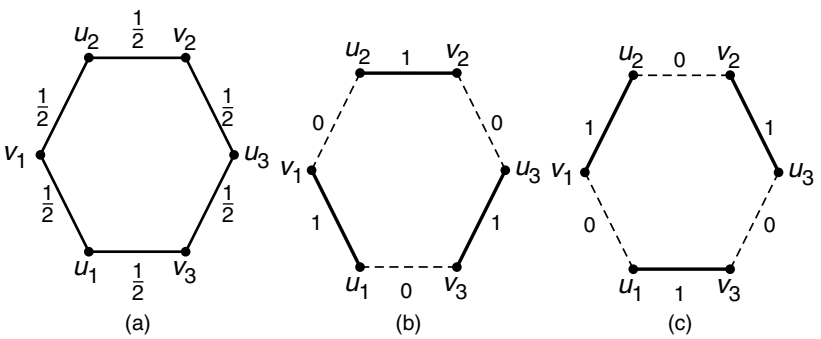

Figure 1.2 (a) The fractional solution which places $\frac{1}{2}$ on all the edges is an optimal fractional solution but not an extreme point solution. The fractional solution in (a) is the convex combination of the integral solutions in (b) and (c).

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 $0 < x_{uv} < 1$ for every edge $\{u,v\} \in E$. We use this assumption to derive a lower bound on the number of variables of the linear program. Let n be the number of 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 preceding outline in Chapter 3.

1.2.2 Approximation algorithms for NP-hard problems

The preceding framework can be naturally adapted to provide an approximation algorithm via the iterative method. In particular, for this, the preceding iterative algorithm 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 β. If there is a constraint $\sum_i a_i x_i \leq b$ such that $\sum_i a_i \leq b + \beta$, then remove the constraint in the residual formulation.

For the bipartite matching problem, we will see how the iterative algorithm presented here 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 next.

(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 [67]. 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 case, 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 iterative algorithm. This involves arguing the following two facts. We establish, first, 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 a 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 a 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 preceding framework can be naturally adapted to provide an approximation algorithm via the iterative method. In particular, recall that this iterative algorithm 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 α for covering problems addressed using this framework.

(ii) **Relaxation:** Fix a threshold β. If there is a constraint $\sum_i a_i x_i \leq b$ such that $\sum_i a_i \leq b + \beta$ then remove the constraint 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 β. 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 preceding 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 Context and applications of iterative rounding

One goal in presenting the collections of results in this book is to convince the reader that iterative rounding is an effective tool in proving results in optimization. As with any tool, a key question is: When is this tool applicable and what are the alternates?

The iterative method for exact optimization used a rank-based argument of the sparsity of the solution to argue integrality of a proposed linear programming formulation of the underlying problem. In Section 13.2, we detail the earliest application we know of this method to prove Steinitz's result on rearrangements.

As we mentioned in the introduction, the iterative method for approximation algorithms was introduced in the work of Jain on the survivable network design problem. For this minimum-cost subgraph selection problem, Jain formulated a covering linear program and showed how any extreme point always has a positive variable of value at least half; he did this by using the sparsity of the extreme point solution, which followed from a rank-based argument. In this context, the iterative method is a specific version of the *deterministic rounding* paradigm applied to LP relaxations for NP-hard problems. Thus, it fits in the broader context of a variety of other LP rounding methods for the design of approximation algorithms including randomized rounding, primal-dual methods, and Lagrangean relaxations. Among these methods, iterative rounding is particularly applicable in solving multiobjective problems where a base problem is complicated by more than one objective function: Examples include the bipartite matching problem complicated by additional load constraints at each node to give the NP-hard generalized assignment problem, or the minimum spanning tree (MST) problem complicated by degree constraints on nodes gives the NP-hard bounded-degree MST problem. An understanding of the iterative method applied to the base problem is then a useful guide to extending its application to the constrained multiobjective versions.

1.5 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 extreme point solutions. Finally, we 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 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 that 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 [46]. 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. 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.

1.6 Notes

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 pioneering work by Jack Edmonds [34]; we refer the reader to the book [121] and the historical survey [119] by Schrijver for an encyclopedic treatment of this subject.

Two closely related methods for proving integrality of polyhedra that are widely covered in Schrijver's book deserve mention: Total Unimodularity (TU) and Total Dual Integrality (TDI). Informally, TU matrices are constraint matrices such that for integral right-hand sides, the linear programming relaxations provide integral solutions (whenever the solutions exist and are finite). Alternately, using the relation between extreme points solutions and basic feasible solutions to LPs developed in the next chapter, these matrices are those for which every square submatrix has determinant value zero, plus one or minus one. The class of Network matrices that we will study in Chapter 8 is an important example of such TU matrices. Total Dual Integrality involves both the constraint matrix and the right-hand side: A system of inequalities defined by a constraint matrix and right-hand side vector is TDI if, for all integer objective coefficients, the dual program has an integral solution (whenever it exists and is finite). If a system is TDI for an integral right-hand side, then the polyhedon described by the system is integral hence giving another way of providing characterizations of integral solutions to combinatorial optimization problems. A popular example of an integral characterization that arises from a TDI system is the description of matchings in general graphs that we develop using our alternate iterative method in Chapter 9.

An implicit use of the iterative method is found in the alternate proof of Steinitz's theorem due to Grinberg and Sevastyanov [10, 65, 127]. Earlier uses of the iterative relaxation method can be traced back to the proof of a discrepancy theorem by Beck and Fiala [14] and the approximation algorithm for the bin