CHAPTER

3

APPROXIMATING COVERING AND PACKING PROBLEMS: SET COVER, VERTEX COVER, INDEPENDENT SET AND RELATED PROBLEMS

Dorit S. Hochbaum

This chapter presents the developments that lead to the use of linear programming formulation as an essential approximation tool. These tools were initially developed for the set cover—the most important and general covering problem—and the vertex cover problem. We describe here the use of linear programs' optimal solution and feasible dual solution for effective approximations for the set cover problem and several closely related problem of covering and packing type. The problems analyzed here in detail include (in addition to the set cover and vertex cover problems), the independent set problem, the multicover problem, the set packing problem, the maximum coverage problem, and the problem of integer programming that extends the vertex cover and independent set problems. We also analyze the properties of the greedy algorithm for covering problems.

3.1 INTRODUCTION

One of the most important tools to have emerged in the design of approximation algorithms is the use of linear programming relaxation of the problem and its dual. We trace

the history and development of this approach as it evolved for the set cover, set packing, and related problems.

The problems discussed in this chapter include the vertex cover problem and the independent set problem, the set cover problem, the multicover problem, and the set packing problem. In addition, we address the problem of maximum covering of elements with minimum number of sets and the problem of integer programs with two variables per inequality. The latter problem is neither a covering nor a packing problem; yet it engulfs in its structure the very properties of the vertex cover problem and the independent set problem that are instrumental in making improved approximation algorithms possible. The analysis of integer programs with two variables per inequality deepens our insights for the reasons that make vertex cover and related problems approximable within a factor of 2 or better.

Other forms of covering and packing problems that are more structured can have improved approximations exploiting the special structure. Some Euclidean covering and packing approximations are described in Chapter 8 and Section 9.3.3. Some network design problems and connectivity problems are possible to present as covering problems, and then techniques that extend those in this chapter are applicable (see Chapter 4 and Section 9.2.1). Also, the vertex cover and independent set problems defined on special classes of graphs have better approximations than the general cases. These special cases are described in detail in Section 3.7.

The linear programming (LP) relaxation plays an important role for all these problems. All known approximation algorithms for the set cover problem [Chv79] [Hoc82] use the (weak) duality theorem of linear programming and the superoptimality of the linear programming relaxation. For the vertex cover, the best known approximation algorithms are provided by independent set and integer programs with two variables per inequality; i.e., the preprocessing technique based on the properties of the linear programming solutions [Hoc83] [HMNT93]. We start by defining the problems discussed in this chapter and how they are related.

3.1.1 DEFINITIONS, FORMULATIONS AND APPLICATIONS

A *vertex cover* in an undirected graph $G = (V, E)$ is a set of vertices C such that each edge of G has at least one endpoint in C. The *vertex cover problem* is the problem of finding a cover of the smallest weight in a graph whose vertices carry positive weights. This problem is known to be *NP*-complete even when the input is restricted to planar cubic graphs with unit weights [GJS76]. An *independent set* in a graph is a set of pairwise nonadjacent vertices (also referred to as *vertex packing*). The largest weight independent set is the complement of the smallest weight vertex cover.

A natural integer programming formulation of the vertex cover problem with node weights w_j for $j \in V$, $|V| = n$, is,

$$
\text{(VC)} \qquad \begin{array}{lll} \text{Min} & \sum_{j=1}^{n} w_j x_j & \\ \text{subject to} & x_i + x_j \geq 1 & \text{(for every edge } (i, j) \text{ in the graph)} \\ & 0 \leq x_j \leq 1 & (j = 1, \ldots, n) \\ & x_j \text{ integer} & (j = 1, \ldots, n). \end{array}
$$

The formulation of the independent set problem (IS) is similar, with "Max" replacing "Min" and with the direction of the first set of inequalities reversed. The independent set problem is also known as the "vertex packing" problem or as the "stable set" problem. To illustrate the problems, consider the graph in Figure 3.1 with all weights equal to 1. The minimum vertex cover is the set of nodes $\{1, 3, 4, 6\}$ and the maximum independent set is the set of nodes $\{2, 5\}$. The linear programming relaxation of (VC) is obtained by removing the integrality constraints on the x_j's.

Among the multiple applications of the vertex cover and the independent set problems are finding nonconflicting schedules. Then assigning the smallest number of watch guards located at vertices so that all links (edges) have at least one guard surveying them.

A general packing problem is the *set packing problem*. Here the goal is to find the maximum weight collection of sets so that no two overlap:

maximize $\{\mathbf{wx} | A \cdot \mathbf{x} \leq \mathbf{e}\}$ for $\mathbf{x}$ binary, $\mathbf{e}$ a column vector of ones and A a zero-one matrix.

This problem can be represented as the independent set problem by constructing a graph (called the derived graph) whose vertices are columns of A, and two such vertices being adjacent if the two columns have a nonzero dot product. The independent set problem is also a special case of the set packing problem, and hence the two problems are polynomially equivalent. As such, any result for the independent set problem is also applicable to the set packing problem. Therefore the set packing problem will not be discussed here separately.

The vertex cover problem is a special case of the set cover problem. Given a set I of m elements to be covered and a collection of sets $S_j \in I$, $j \in J = \{1, \ldots, n\}$. Each set has weight w_j associated with it. The characteristic vector of set S_j is the $0-1$ vector $\{a_{ij}\}_{i=1}^{m}$. The *set cover problem* is to identify the smallest weight collection of sets so that all elements of I are included in their union (or "covered"): minimize $\{\mathbf{wx} | A \cdot \mathbf{x} \geq \mathbf{e}\}$ for $\mathbf{x}$ binary. A vertex cover problem is a set cover problem where each element can be covered by exactly two sets. These two sets correspond to the endpoints of an edge in the graph. Unlike the set packing problem which is equivalent to the vertex packing problem, the set cover problem is a strict generalization of the vertex cover problem, and the two problems are distinguished by the quality of approximation algorithms that can be devised for them.

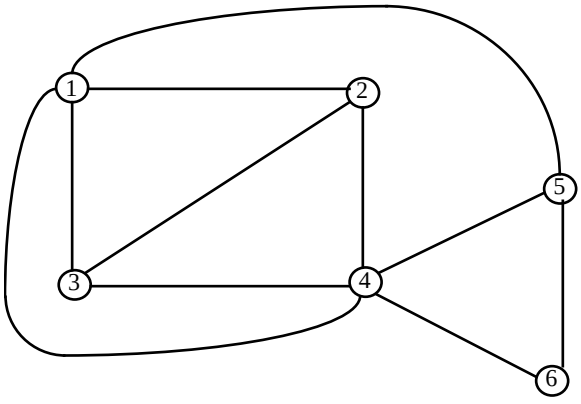

FIGURE 3.1

The set cover problem has applications in diverse contexts such as efficient testing, statistical design of experiments, [FNT74], and crew scheduling for airlines [MMK79]. It also arises as a subproblem of many integer programming problems. For surveys on the set cover problems see Garfinkel and Nemhauser [GN72], Christofides and Korman [CK75], Balas and Padberg [BP76], Padberg [Pad79], and an annotated bibliography by Trotter [Tro85].

Some applications of the set cover problem require an extension where each element is to be covered a specified number of times. This extension is called the *multicover* problem. The multicover problem has applications where reliability of coverage requires extra redundancy. Among the applications of the problem are the location of emergency service facilities, communication systems, military applications marketing applications, crew scheduling, and security checking (Van Slyke 81 [VS81]). In the formulation of the multicover problem, each element i is to be covered at least b_i times.

$$\text{(MC)} \qquad \begin{array}{ll} \text{Min} & \sum_{j=1}^{n} w_j x_j \\ \text{subject to} & \sum_{j=1}^{n} a_{ij} x_j \geq b_i \quad (\text{for } i = 1, \ldots, m\ ) \\ & 0 \leq x_j \leq 1 \quad (j = 1, \ldots, n) \\ & x_j \ \text{ integer} \quad (j = 1, \ldots, n). \end{array}$$

When the amount of required coverage $b_i = 1$ for all i, the multicover problem reduces to the set cover problem (SC).

The maximum coverage problem generalizes the set cover problem and the multicover problem. Here, instead of seeking the smallest number of sets that cover all elements, we seek the largest number of elements (accounting for their multiplicities) that can be covered by a prespecified number of sets, k. When this largest number is m—the total number of elements to be covered—the solution is also a set cover. The maximum coverage problem is also defined in a weighted context: find the largest number of elements that can be covered by sets of total weight not exceeding W—the budget limit.

Both formulations of the vertex cover and independent set problems have two variables per inequality. Another problem that is formulated as integer programming optimization with two variables per inequality is the problem of minimizing the weight of true variables in a 2-satisfiability truth assignment. In the 2-SAT problem we are given a collection of clauses in conjunctive normal form (CNF) of length 2 each, where each variable has a certain weight associated with setting it to True. The vertex cover problem could be viewed as 2-SAT with no negation of variables. As such, integer programming with two variables per inequality, IP2, captures in its structure a number of other problems. Indeed, as we shall see, much of the insight about these three problems can be derived from the analysis of IP2. The formulation of IP2 is,

$$\text{(IP2)} \qquad \begin{array}{ll} \text{Min} & \sum_{j=1}^{n} w_j x_j \\ \text{subject to} & a_i x_{j_i} + b_i x_{k_i} \geq c_i \quad (\text{for } i = 1, \ldots, m\ ) \\ & 0 \leq x_j \leq u_j \quad (j = 1, \ldots, n) \\ & x_j \ \text{ integer} \quad (j = 1, \ldots, n), \end{array}$$

where $1 \leq j_i, k_i \leq n$, $w_i \geq 0$ $(i = 1, \ldots, n)$, and all the coefficients are integer. We denote the largest upper bound by $U = \max_{j=1,\ldots,n} u_j$.

IP2 with $a_i = b_i = c_i = u_i = 1$ is the vertex cover problem. With $u_i = 1$ and $a_i, b_i, c_i \in \{-1, 0, 1\}$ IP2 is the 2-SAT problem.

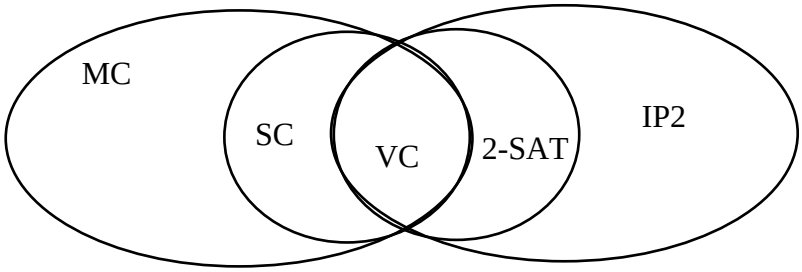

FIGURE 3.2

Figure 3.2 depicts schematically the relationship between vertex cover and the other minimization problems.

3.1.2 LOWER BOUNDS ON APPROXIMATIONS

All problems discussed here are Max-$\mathcal{SNP}$-complete, which means that there are no polynomial approximation schemes, PASs, for these problems unless $NP = P$. Stronger lower bounds have been derived for these problems (and are getting improved continuously). The strongest lower bounds at the time of this writing are:

- For vertex cover problem there is no δ-approximation for $\delta < 16/15$ unless $NP = P$, [BGS95].
- The independent set problem is equivalent to the maximum clique problem on the complement graph. The maximum clique problem has been studied extensively for lower bounds, and there are extremely large (and thus discouraging) lower bounds that apply also to the independent set problem. The current champion lower bound is $\sqrt{n}$ [Håstad, private communication] meaning that we cannot guarantee an approximation factor $n^{\frac{1}{2}-\delta}$ for any positive δ unless $NP = P$.
- There have been some constant lower bounds proved for approximating set cover that hold unless $NP = P$. Stronger bounds were proved under the assumption that $NP \neq DTIME(n^{O(\log\log n)})$. In other words, if NP problems are not solvable in time that is quasi-polynomial, or exponential in $\log\log n$, then the lower bound holds. A lower bound proved recently by Feige [F95] is $(1-\epsilon)\ln n$ provided that $NP \neq DTIME(n^{O(\log\log n)})$.

A lower bound for multicover follows from that of set cover, and a lower bound to IP2 follows from the bound for vertex cover.

When considering these lower bounds one has to keep in mind that these are worst case lower bounds. Indeed, there are approximation algorithms for set cover instances

that have small set sizes or have small coverage duplicity (the number of sets covering a given element), that have performance better than the lower bound. For instance, the approximation factor for an independent set on bounded degree graphs is substantially better than the lower bound of $\sqrt{n}$ implies, and similarly for many other special classes of problems demonstrated in this chapter. Table 3.1 summarizes such results.

3.1.3 OVERVIEW OF CHAPTER

The chapter is arranged in chronological order of developments—with some minor exceptions. We begin with a discussion of the set cover problem and the greedy algorithm which was the first approximation algorithm devised for it. The analysis of the greedy was the first use of linear programming duality in approximations. We then present the Linear Programming (LP) approximation algorithm that mades use of the dual optimal solution; then the dual-feasible algorithm making use of a dual solution that is only feasible rather than optimal, and finally using other relaxations of the set cover problem that lead to a variety of dual-feasible algorithms. These are applicable to the set cover problem, and some are only applicable to its special case—the vertex cover problem. We then extend the analysis of the linear programming algorithm and the dual-feasible algorithm to the multicover problem.

Next, we demonstrate the value of the *optimal* dual solution in a preprocessing approach that yields improved approximation bounds to the vertex cover or the independent set problems. In this section we describe a large number of special classes of these problems along with the improved approximation bounds. All known approximation algorithms to date for these problems are then summarized in Table 3.1.

Section 3.8 investigates the nature of the factor of 2 approximation for the vertex cover problem (which we also conjecture to be best possible), and describes how the ideas of preprocessing and the use of the optimal linear programming solution apply also in the more general set up of integer programming with two variables per inequality.

Finally, we discuss the performance of the greedy algorithm for the maximum coverage problem, which is an extension of the set cover problem (as a decision problem). This problem is of particular interest because of the analysis of the generic type of greedy algorithm involved.

The notation used in this chapter includes bold fonts for vectors; $\mathbf{e}$ denotes the vector of all 1's, and $\mathbf{e}_i$ denotes the vector of all 0's except for a 1 in the i^{th} position.

3.2 THE GREEDY ALGORITHM FOR THE SET COVER PROBLEM

A greedy algorithm is the most natural heuristic for set cover. It works by selecting one set at a time that covers the most elements among the uncovered ones. Johnson and

Lovász ([Joh74], [Lov75]) were the first to demonstrate that the greedy algorithm is a $\mathcal{H}(d)$- approximation algorithm for the unweighted set cover problem, where $\mathcal{H}(d) = \sum_{i=1}^{d} \frac{1}{i}$ and d is the size of the largest set. $\mathcal{H}(d)$ is bounded by $1 + \log d$.

Chvátal [Chv79] extended the applicability of the greedy to the weighted set cover. This version of greedy selects a set with the minimum ratio of weight to remaining coverage. Chvátal proved that this greedy algorithm is still a $\mathcal{H}(d)$-approximation algorithm. Although the algorithm is easily stated, its analysis is far from trivial. That analysis is particularly instructive as it introduces the use of linear programming duality in approximations, which we will present next. The formal statement of the greedy is,

THE GREEDY ALGORITHM [CHVÁTAL]

Step 0: Set $C^G = \emptyset$; $S_j^1 = S_j$, $j \in J$; $I = \{1, \ldots, m\}$; $k = 0$.
Step 1: $k \leftarrow k+1$. Select a set S_{j_k}, such that $\frac{w_{j_k}}{|S_{j_k}^k|} = \min_{j \in J} \frac{w_j}{|S_j^k|}$.
Step 2: Set $C^G \leftarrow C^G \cup \{j_k\}$ and $S_j^{k+1} = S_j^k \setminus S_{j_k}^k$, $j \in J$, $I \leftarrow I \setminus S_{j_k}^k$.
Step 3: If $I = \emptyset$, stop and output cover C^G. Else, go to Step 1.

Consider the linear programming relaxation of (SC), with the upper bound $x_j \leq 1$ constraints omitted (an optimal solution will satisfy those constraints automatically). The dual problem is

$$\text{(SC-dual)} \qquad \begin{array}{lll} \text{Max} & \sum_{i=1}^{m} y_i & \\ \text{subject to} & \sum_{i=1}^{m} a_{ij} y_i \leq w_j & (\text{for } j = 1, \ldots, n\,) \\ & y_i \geq 0 & (i = 1, \ldots, m). \end{array}$$

The analysis of greedy relies on allocating the weights of the set selected by the greedy heuristic to the elements covered, and interpreting those as a form of dual, not quite feasible, solution.

THEOREM 3.1 The greedy heuristic is a $\mathcal{H}(d)$-approximation algorithm.

Proof. To prove the desired result, it suffices to show that for *any* cover C, indicated by the characteristic vector $\{x_j\}$, and a cover delivered by the greedy C^G,

$$\sum_{j \in C} \mathcal{H}(d) w_j = \sum_{j=1}^{n} \mathcal{H}(d) w_j x_j \geq \sum_{j \in C^G} w_j \,. \tag{3.1}$$

Applying this inequality to C^*, the optimal cover, yields that the value of the solution delivered by the greedy is at most $\mathcal{H}(d)$ times the value of the optimal solution. To prove (3.1), it is sufficient to find an "almost feasible" dual solution $\mathbf{y}$ such that,

$$\sum_{i=1}^{m} a_{ij} y_i \leq \mathcal{H}(|S_j|) w_j \quad j = 1, \ldots, n \tag{3.2}$$

and so that the weight of the sets selected is accounted for by **y**,

$$\sum_{i=1}^{m} y_i = \sum_{j \in C^G} w_j. \tag{3.3}$$

Such **y** satisfying these inequalities is feasible within a factor of $\mathcal{H}(d)$, and it satisfies (3.1) since,

$$\sum_{j=1}^{n} \mathcal{H}(d) w_j x_j \overset{(3.2)}{\geq} \sum_{j=1}^{n} (\sum_{i=1}^{m} a_{ij} y_i) x_j = \sum_{i=1}^{m} (\sum_{j=1}^{n} a_{ij} x_j) y_i \geq \sum_{i=1}^{m} y_i \overset{(3.3)}{=} \sum_{j \in C^G} w_j. \tag{3.4}$$

Let S_j^k be the set S_j with the remaining elements at the beginning of iteration k, and its size, $|S_j^k| = s_j^k$. The dual vector **y** that will satisfy (3.2) and (3.3) has for y_i the average price paid by the greedy to cover an element i. Whenever a set is selected, its weight is divided evenly among the elements it has newly covered, $y_i = \frac{w_k}{s_k^k}$.

Let the sets greedy selects in the first k iterations be $\{1, 2, \ldots, k\}$. Since k is the index for which the ratio is minimum

$$\frac{w_k}{s_k^k} \leq \frac{w_j}{s_j^k} \qquad \forall j. \tag{3.5}$$

Assume that there are t iterations altogether. Then, $\sum_{j \in C^G} w_j = \sum_{j=1}^{t} w_j$. Each element $i \in I$ belongs to one set $S_k^k, k = 1, \ldots, t$, so for $i \in S_k^k$, $y_i = \dfrac{w_k}{s_k^k}$. (3.3) now follows as,

$$\sum_{i=1}^{m} y_i = \sum_{k=1}^{t} \sum_{i \in S_k^k} y_i = \sum_{k=1}^{t} s_k^k (\frac{w_k}{s_k^k}) = \sum_{k=1}^{t} w_k \ .$$

To prove (3.2) observe that, $S_j \cap S_k^k = S_j^k \setminus S_j^{k+1}$ and $I = \bigcup_{k=1}^{t} S_k^k$. Hence,

$$\sum_{i=1}^{m} a_{ij} y_i = \sum_{k=1}^{t} \sum_{i \in S_j \cap S_k^k} y_i = \sum_{k=1}^{t} \sum_{i \in S_j^k \setminus S_j^{k+1}} y_i = \sum_{k=1}^{t} \left(s_j^k - s_j^{k+1} \right) \frac{w_k}{s_k^k} \ .$$

For a given set S_j, let p be the largest index such that $s_j^p > 0$, then

$$\sum_{i=1}^{m} a_{ij} y_i = \sum_{k=1}^{p} (s_j^k - s_j^{k+1}) \frac{w_k}{s_k^k} \overset{(3.5)}{\leq} w_j \sum_{k=1}^{p} \frac{s_j^k - s_j^{k+1}}{s_j^k} \ .$$

We now use the inequality $\frac{s_j^k - s_j^{k+1}}{s_j^k} \leq \mathcal{H}(s_j^k) - \mathcal{H}(s_j^{k+1})$ to establish,

$$\sum_{i=1}^{m} a_{ij} y_i \leq w_j \sum_{k=1}^{p} \left(\mathcal{H}(s_j^k) - \mathcal{H}(s_j^{k+1}) \right) \leq w_j \mathcal{H}(s_j^1) \ .$$

■

The greedy algorithm is thus an $O(\log n)$-approximation algorithm for any set cover. This matches the recently proved lower bound for approximating the set cover [F95]. Still considerably better results are possible for special cases. Consider for in-

stance the performance of the greedy on *high coverage* instances of unweighted set cover.

EXERCISE 3.1 [KZ95]: Let an unweighted set cover instance be of α-coverage if every element of I belongs to at least $\alpha|J|$ sets. Then the greedy delivers a solution of size $\log_{\frac{1}{1-\alpha}} n$ for instances of α-coverage.

As a corollary an enumeration algorithm solves the α-coverage instance in $O(m^{O(\log n)})$ steps. The high coverage problem—where α is close to 1—is thus unlikely to be *NP*-hard.

3.3 THE LP-ALGORITHM FOR SET COVER

A different approximation algorithm—which is duality based - was devised for the set cover problem by Hochbaum [Hoc82]. This was motivated by an approximation to the *unweighted* vertex cover problem by Gavril (reported as private communication in [GJ79]). Gavril's algorithm is based on the idea of solving for a maximal (not necessarily maximum) matching, and taking both endpoints of the edges in the matching. The number of edges in the maximal matching $|M|$ is a lower bound on the optimum $|VC^*|$. This is because a single vertex cannot cover two edges in M. On the other hand, if we pick both endpoints of each matched edge we get a feasible cover, VC^M, as otherwise there would be an edge with both endpoints unmatched. Therefore this edge could be added to the matching M—contradicting its maximality. These two statements lead to the inequalities,

$$|VC^M| = 2|M| \leq 2|VC^*|. \tag{3.6}$$

Hence, this cover is at most twice the optimal cover.

The extension of this idea to the weighted case and to the set cover problem was inspired by an alternative way of viewing the maximal matching algorithm as a feasible dual solution. To see that, consider the dual of the linear programming relaxation of the unweighted vertex cover problem.

$$\text{(VC-dual)} \qquad \begin{array}{lll} \text{Max} & \sum_{(i,j)\in E} y_{ij} & \\ \text{subject to} & \sum_{(i,j)\in E} y_{ij} \leq 1 & \text{for } j = 1,\ldots,n \\ & y_{ij} \geq 0, & \forall\ (i,j) \in E. \end{array}$$

An integer feasible solution to (VC-dual) is a matching in the graph. Consider a feasible solution to the dual, $\bar{\mathbf{y}}$, and let $\bar{x}_j = 1$ whenever the dual constraint is binding, $\sum_{(i,j)\in E} \bar{y}_{ij} = 1$. We show that the solution $\bar{\mathbf{x}}$ is a feasible vertex cover. To that end, we introduce the concept of maximality: a feasible solution to (VC-dual), $\bar{\mathbf{y}}$, is said to be *maximal* if there is no feasible solution $\mathbf{y}$ such that $y_{ij} \geq \bar{y}_{ij}$ and $\sum_{(i,j)\in E} y_{ij} > \sum_{(i,j)\in E} \bar{y}_{ij}$.

LEMMA 3.1 Let $\bar{\mathbf{y}}$ be a maximal feasible solution to $(VC-dual)$. Then the set $VC = \{i \mid \sum_{(i,j)\in E} \bar{y}_{ij} = 1\}$ is a feasible solution to (VC).

Proof. Suppose that VC is not a feasible cover. Then there is an edge (u,v) which is uncovered, i.e., $\sum_{(u,j)\in E} y_{uj} < 1$ and, $\sum_{(v,j)\in E} y_{vj} < 1$.

Let $\delta = \text{Min}\{1-\sum_{(u,j)\in E} y_{uj}, 1-\sum_{(v,j)\in E} y_{vj}\}$. Then the vector, $\mathbf{y} = \bar{\mathbf{y}} + \delta \cdot \mathbf{e}_{uv}$ (for $\mathbf{e}_{uv}$ denoting the vector of all zeros except for a 1 in the uv entry), is a feasible solution satisfying $y_{ij} \geq \bar{y}_{ij}$ and $\sum_{(i,j)\in E} y_{ij} > \sum_{(i,j)\in E} \bar{y}_{ij}$. This contradicts the maximality of $\bar{\mathbf{y}}$. Hence, every edge must be covered by VC and VC is therefore a feasible cover. ■

Consider now the generalization of this approach for the set cover problem.

DEFINITION 3.1 A feasible solution to (SC-dual) $\bar{\mathbf{y}}$ is said to be *maximal* if there is no feasible solution $\mathbf{y}$ such that $y_i \geq \bar{y}_i$ and $\sum_{i=1}^{n} y_i > \sum_{i=1}^{n} \bar{y}_i$.

THE LP-ALGORITHM [HOCHBAUM]

Step 1: Find a maximal dual feasible solution for (SC), $\bar{\mathbf{y}}$.
Step 2: Output the cover $C^H = \{j \mid \sum_{i=1}^{n} a_{ij}\bar{y}_i = w_j\}$.

LEMMA 3.2 C^H is a feasible solution to (SC).

Proof. The proof is an obvious extension of Lemma 3.1: Consider an element q that is not covered. Let $\delta = \text{Min}_{j|q\in S_j}\{w_j - \sum_{j=1}^{n} a_{ij}\bar{y}_i\} > 0$. Then the vector, $\mathbf{y} = \bar{\mathbf{y}} + \delta \cdot \mathbf{e}_q$, is a feasible solution contradicting the maximality of $\bar{\mathbf{y}}$. ■

Let C^* be the optimal cover. Define for any set cover C, $w(C) = \sum_{j\in C} w_j$. The following lemma shows that the the LP- algorithm is a $\max_i\{\sum_j a_{ij}\}$- approximation algorithm due to the dual constraint being binding for every $j \in C^H$.

LEMMA 3.3 $w(C^H) \leq \max_i\{\sum_j a_{ij}\} w(C^*)$.

Proof. First, $w(C^H) = \sum_{j\in C^H} w_j = \sum_{j\in C^H}(\sum_{i=1}^{m} a_{ij}y_i)$. Using the weak duality theorem we get that for any solution to the linear programming relaxation, and in particular for the optimal solution $\mathbf{x}^*$,

$$\begin{aligned}\sum_{j\in C^H}(\sum_{i=1}^{m} a_{ij}y_i) &\leq \max_i\{\sum_{j\in C^H} a_{ij}\}\sum_{i=1}^{m} y_i \leq \max_i\{\sum_{j\in C^H} a_{ij}\}\sum_{j=1}^{n} w_j x_j^* \\ &\leq \max_i\{\sum_{j\in C^H} a_{ij}\} w(C^*).\end{aligned}$$

Now the value of the optimal solution to the linear programming relaxation is a lower bound to the optimal integer solution. It follows that,
$w(C^H) \leq \max_i\{\sum_{j\in C^H} a_{ij}\} w(C^*)$. ■

This proves a slightly stronger approximation factor as we can consider the row sums restricted only to those sets in the cover (alternatively the row sums are calculated in the submatrix of the columns in the cover). For instance, if the cover C^H has only one

set covering each element, then it is optimal. Let the maximum number of sets covering an element, $\max_i\{\sum_j a_{ij}\}$ be denoted by p.

An immediate corollary of the p-approximation is that the LP-algorithm is a 2-approximation algorithm for the (weighted) vertex cover problem.

COROLLARY 3.1 The LP-algorithm is a 2-approximation algorithm for the vertex cover problem.

Proof. (VC) is a special case of (SC) with each element—an edge—belonging to precisely two "sets" representing its endpoints. Hence, $\sum_j a_{ej} = 2$ for all edges $e \in E$. ■

The implementation of the LP-algorithm proposed in [Hoc82] used the optimal dual solution as a maximal feasible solution. Still, the role of the dual solution is only to aid the analysis of the algorithm. It need not be generated explicitly by the algorithm:

THE ROUNDING ALGORITHM [HOCHBAUM]

Step 1: Solve optimally the linear programming relaxation of (SC). Let an optimal solution be $\{x_j^*\}$

Step 2: Output the cover $C^H = \{j | x_j^* > 0\}$. Equivalently, set $x_j^H = \lceil x_j^* \rceil$.

The rounding algorithm is indeed a special case of the LP-algorithm as $x_j^* > 0$ implies that the corresponding dual constraint is binding by complementary slackness optimality conditions, $\sum_{i=1}^m a_{ij} y_i = w_j$.

A minor variation of the rounding algorithm is still a p-approximation algorithm. We replace Step 2 by:

Step 2′: Output the cover $C^H = \{j | x_j^* \geq \frac{1}{p}\}$.

The feasibility of this cover is obvious, as in any fractional solution $\mathbf{x}$ corresponding to a cover C, $\sum_{j=1}^n x_j \geq 1$, and there are at most p positive entries per such inequality. So at least one must be at least as large as the average $\frac{1}{p}$. This rounding algorithm (rounding II) will always produce a cover no larger than the rounding algorithm; although it does not offer any advantage in terms of worst case analysis. Moreover, for any cover produced by an approximation algorithm, it is easy to prune it of unnecessary extra sets, and leave a "prime" cover, which is a *minimal* cover. Although a prime cover can only be a better solution, this approach has not provided *guaranteed* tighter approximation factors.

In the next section we see that it is not necessary to compute an *optimal* solution to the linear programming problem. Rather, there are more efficient ways of finding a maximal dual solution, two of which are described in the next section. On the other hand, as discussed in the section on the preprocessing algorithm, 3.7, the linear programming solution carries some extra valuable information for the vertex cover and independent set problems. In addition, for the vertex cover problem it is possible to solve the linear

programming relaxation by applying a max-flow min-cut algorithm which is more efficient than solving the respective linear program.

It is also shown that the LP-algorithm can also be used in the presence of covering matrices with coefficients other than 0 or 1. In Section 3.5 it is demonstrated for a different formulation of the set cover problem. Chapter 4 is devoted entirely to applications of the algorithm of finding maximal dual solutions to a large variety of covering-type problems.

3.4 THE FEASIBLE DUAL APPROACH

Solving the linear programming relaxation of set cover can be done in polynomial time. Yet, much more efficient algorithms are possible that find a feasible and maximal dual solution. The advantage of such algorithms is in the improved complexity. The approximation ratios derived are the same as for the dual optimal solution.

Bar-Yehuda and Even [BYE81] devised an efficient algorithm for identifying a maximal feasible dual solution to be used in the LP-algorithm. The idea is to identify a feasible primal constraint and then to increase its dual variable till at least one of the dual constraints becomes binding.

As before, the derivation of the dual solution is implicit and exists only in order to analyze the approximation factor. This dual information is placed in square brackets in the description of the algorithm to stress that it is not an integral part of the procedure.

THE DUAL-FEASIBLE I ALGORITHM [BAR-YEHUDA AND EVEN]

Step 0: Set $C = \emptyset$; $I = \{1, \ldots, m\}$; $[\mathbf{y} = 0]$.
Step 1: Let $i \in I$. Let $w_{j(i)} = \min_{a_{ij}=1} w_j$. $[y_i = w_{j(i)}]$; $C \leftarrow C \cup \{j(i)\}$.
Step 2: {update} For all j such that $a_{ij} = 1$, $w_j \leftarrow w_j - w_{j(i)}$, $I \leftarrow I \setminus S_j(i)$.
Step 3: If $I = \emptyset$, stop and output cover C. Else, go to Step 1.

Throughout this procedure the updated w_js remain nonnegative. The weights w_j are in fact the reduced costs corresponding to the solution $\mathbf{y}$, and at each iteration they quantify the amount of slack in the dual constraint. In this sense, the algorithm is dual-feasible—throughout the procedure it maintains the feasibility of the dual vector $\mathbf{y}$. For any set added to the cover, the updated value of the reduced cost is 0, i.e. the corresponding dual constraint is binding. The resulting dual vector is maximal since each element belongs to some set in the cover, and therefore to some set with a binding dual constraint. Hence, there is no vector that is larger or equal to $\mathbf{y}$ in all its components which is feasible, unless it is equal to $\mathbf{y}$.

Lemma 3.3 applies to the cover delivered by the dual-feasible I algorithm, namely, dual-feasible I is a p-approximation algorithm to the set cover problem. The

complexity of the dual-feasible algorithm is $O(mn)$, which is linear in the size of the input matrix, and hence a considerable improvement to the complexity of the respective linear program.

As in Corollary 3.1, this algorithm is a 2-approximation to the vertex cover problem. However, as discussed in the next section, there is additional information in the optimal linear programming solution to the vertex cover problem that is lost in the dual-feasible algorithm.

Another variation on the theme of dual feasible solutions was proposed by Clarkson [Clar83] for the vertex cover problem. This algorithm is superficially similar to the greedy algorithm: instead of choosing a vertex based on its minimum weight per edge covered, it is choosing a vertex based on minimum *reduced* weight. Clarkson did not use the concept of duality, and his proof for the bound of 2 is consequently more involved. We adapt this idea for the set cover problem:

THE DUAL-FEASIBLE ALGORITHM II [AFTER CLARKSON]

Step 0: Set $C = \emptyset$; $S_j^1 = S_j$, $j \in J$; $I = \{1, \ldots, m\}$; $k = 0$; [$\mathbf{y} = 0$].

Step 1: $k = k + 1$. Select a set S_{j_k}, such that $\frac{w_{j_k}}{|S_{j_k}^k|} = \min_{j \in J} \frac{w_j}{|S_j^k|}$.

Step 2: {update} Set $C \leftarrow C \cup \{j_k\}$ and $S_j^{k+1} = S_j^k \setminus S_{j_k}^k, \forall j \in J$, $I \leftarrow I \setminus S_{j_k}^k$. $w_j \leftarrow w_j - \frac{w_{j_k}}{|S_{j_k}^k|} \cdot |S_j^k \cap S_{j_k}^k|$. [$y_i = \frac{w_{j_k}}{|S_{j_k}^k|} \forall i \in S_{j_k}^k$.]

Step 3: If $I = \emptyset$, stop and output cover C. Else, go to Step 1.

Dual-feasible II has several interesting aspects. Whenever a set is selected, its corresponding dual constraint becomes binding as each of the elements covered by it is assigned an equal share of the set's (reduced) weight. This symmetry among the elements in the allocation of the dual weights makes the algorithm particularly amenable to parallel and distributed implementations. Indeed Khuller, Vishkin, and Young, [KVY94], devised a parallel algorithm that runs in time $O(\log^2 m \log \frac{1}{\epsilon})$, and produces a solution that is at most $\frac{p}{(1-\epsilon)}$ times the optimum.

In the next section we demonstrate how Dual-feasible II could have been conceived from a different formulation of the set cover problem. This underlies the connection between formulations and algorithms.

3.5 USING OTHER RELAXATIONS TO DERIVE DUAL FEASIBLE SOLUTIONS

Imagine an alternative formulation of the set cover problem that has many additional constraints compared to the standard formulation. Let $\bar{C}$ be any feasible set cover and any $S \subseteq I = \{1, \ldots, m\}$. Then, obviously $\sum_{j \in \bar{C}} |S_j \cap S| \geq |S|$. This leads to a new

formulation with the following LP relaxation of the set cover problem:

$$\begin{array}{ll} \text{Min} & \sum_{j=1}^{n} w_j x_j \\ \text{subject to} & \sum_{j=1}^{n} |S_j \cap S| x_j \geq |S| \quad \forall S \subseteq I \\ & x_j \geq 0, \quad j \in J. \end{array}$$

Observe that this formulation contains all the constraints of (SC) for $S = i$, $i \in I$. All the additional constraints are redundant. The coefficients in the constraint matrix are no longer 0 and 1 as before. The dual to this relaxation is

$$\begin{array}{ll} \text{Max} & \sum_{S} |S| y_S \\ \text{subject to} & \sum_{S} |S_j \cap S| y_S \leq w_j, \quad j \in J \\ & y_S \geq 0 \qquad S \subseteq I. \end{array}$$

Consider now the dual feasible algorithm applied to this formulation. For every violated primal constraint (uncovered element), we increase the corresponding y_S proportionally to its coefficient in all dual constraints until at least one becomes binding. This means setting

$$y_S = \min_{j | S_j \cap S \neq \emptyset} \frac{w_j}{|S_j \cap S|}.$$

In particular, we may choose $S = I$ and add the set S_k for which the minimum is attained to the cover; update $I \leftarrow I \setminus S_k$ and repeat.

Notice that this algorithm is precisely Dual-feasible II. To see that the same p-approximation follows notice that for every pair of feasible covers C and $\bar{C}$,

$$\sum_{j \in C} |S_j \cap S| \leq p|S| \leq p \sum_{j \in C} |S_j \cap S|.$$

Let the optimal integer solution be $|SC^*|$. We now have similar inequalities as before,

$$\sum_{j \in C} w_j = \sum_{j \in C} \sum_{S} |S_j \cap S| y_S = \sum_{S} \sum_{j \in C} |S_j \cap S| y_S \leq p \sum_{S} |S| y_S \leq p|SC^*|.$$

This type of approach that uses alternative formulations has lead to considerably better approximations for specific types of covering problems and network design problems as described in Chapter 4. For an alternative 2-approximation for the vertex cover problem, see Section 9.2.1.

3.6 APPROXIMATING THE MULTICOVER PROBLEM

In this section we present variations of the LP-algorithm, the rounding algorithm, and the dual-feasible algorithm that also work for the multicover problem (MC). The description of the dual-feasible algorithm and its analysis are from [HH86]. Consider the linear

programming relaxation of the multicover problem and its dual:

$$\begin{array}{lll} & MC^* = \text{ Min} & \sum_{j=1}^{n} w_j x_j \\ \text{(MCR)} & \text{subject to} & \sum_{j=1}^{n} a_{ij} x_j \geq b_i \quad (\text{for } i = 1, \ldots, m\,) \\ & & 0 \leq x_j \leq 1 \quad (j = 1, \ldots, n) \end{array}$$

MC* obviously bounds from below the optimal value to the multicover problem. The dual to the linear programming relaxation above reads:

$$\begin{array}{lll} & \text{Max} & \sum_{i=1}^{m} b_i y_i - \sum_{j=1}^{n} v_j \\ \text{(MC-dual)} & \text{subject to} & \sum_{i=1}^{m} a_{ij} y_i - v_j \leq w_j \quad (\text{for } j = 1, \ldots, n\,) \\ & & y_i, v_j \geq 0 \quad (i = 1, 2, \ldots, m, j = 1, \ldots, n) \end{array}$$

Here a feasible dual solution $(\bar{\mathbf{y}}, \bar{\mathbf{v}})$ is called *maximal* if it satisfies:

(i) There is no other feasible solution (y, v) such that $y_i \geq \bar{y_i}$, $v_i \geq \bar{v_i}$ and $\sum_{i=1}^{m} b_i y_i - \sum_{j=1}^{n} v_j > \sum_{i=1}^{m} b_i \bar{y_i} - \sum_{j=1}^{n} \bar{v_j}$.

(ii) $\bar{v_j} = 0$ whenever $\sum_{i=1}^{m} a_{ij} y_i < w_j$.

(iii) $\sum_{i=1}^{m} \bar{y_i} \leq \sum_{i=1}^{m} b_i \bar{y_i} - \sum_{j=1}^{n} \bar{v_j}$

With this definition of maximality, the following LP-algorithm works as a p-approximation algorithm, where $p = \max_i\{\sum_j a_{ij}\}$.

THE LP-ALGORITHM FOR MULTICOVER

Step 1: Find a maximal dual feasible solution for (MC), $\bar{\mathbf{y}}, \bar{\mathbf{v}}$.

Step 2: Output the cover $C^H = \{j \mid \sum_{i=1}^{n} a_{ij} \bar{y_i} - \bar{v_j} = w_j\}$.

LEMMA 3.4 The LP-algorithm is a p-approximation algorithm for (MC).

Proof. First, we establish that C^H is a feasible multicover. Consider an uncovered element (row) q. Notice that for the problem to be feasible, every row i needs at least b_i sets covering it. Let $\delta = \text{ Min }_{j|q \in S_j}\{\delta_j = w_j - \sum_{j=1}^{n} a_{ij} \bar{y_i} \text{ and } \delta_j > 0\}$. Since there must be at least b_q sets that q belongs to, and at most $b_q - 1$ of them are in C^H, it follows that δ is well defined. Now set, $\mathbf{y} = \bar{\mathbf{y}} + \delta \cdot \mathbf{e}_q$ and $\mathbf{v} = \bar{\mathbf{v}} + \sum_{j \in C^H | q \in S_j} \delta \mathbf{e}_j$.

It is easy to verify that the vector $(\mathbf{y}, \mathbf{v})$ is a feasible solution, thus contradicting property (i) of the maximality of $(\bar{\mathbf{y}}, \bar{\mathbf{v}})$. This is because the first term has at least increased by $b_q \delta$ whereas the second term has at most increased by $(b_q - 1)\delta$, thus contributing to a net increase of the objective function by at least δ.

Now $(\bar{\mathbf{y}}, \bar{\mathbf{v}})$ is a feasible dual solution, hence the weak duality theorem applies:

$$\sum_{i \in I} b_i \bar{y_i} \leq \min\left(\sum_{j \in J} w_j x_j\right) + \sum_{j \in J} \bar{v_j} \leq \text{MC}^* + \sum_{j \in J} \bar{v_j}.$$

From that and property (iii),

$$\sum_{i \in I} y_i \leq \text{MC}^*. \tag{3.7}$$

Using the construction of C^H:

$$\sum_{j\in C^H} w_j + \sum_{j\in C^H} v_j = \sum_{j\in C^H}\sum_{i\in I} a_{ij}\bar{y_i} = \sum_{i\in I}\left(\sum_{j\in C^H} a_{ij}\right)\bar{y_i}$$
$$\leq \left(\max_{i\in I}\sum_{j\in C^H} a_{ij}\right)\cdot\sum_{i\in I}\bar{y_i} \leq p\cdot \mathrm{MC}^*.$$

The last inequality follows from 3.7 and the definition of p. Recalling that the $\bar{v_j}$'s are nonnegative and that $\mathrm{MC}^* \leq w(C^*)$ where $w(C^*)$ is the value of the optimal integer solution, we derive the stated result. ■

A rounding algorithm is also a p-approximation algorithm. It offers the advantage of a smaller weight multicover.

THE ROUNDING ALGORITHM [HALL AND HOCHBAUM]

Step 1: Solve the linear programming relaxation of (MC) optimally. Let an optimal solution be $\{x_j^*\}$

Step 2: Output the cover $C^H = \{j | x_j^* \geq \frac{1}{p}\}$.

The feasibility of this cover is obvious, as in any fractional solution **x** corresponding to a cover C, $\sum_{j=1}^n a_{ij}x_j \geq b_i$. So at least b_i entries must be at least as large as the average $\frac{1}{p}$.

Next we present a dual-feasible algorithm that delivers a p-approximate solution to the multicovering problem. This algorithm has a better complexity than the one required to solve the relaxation optimally.

The input to the algorithm is the matrix A and the vectors **b** and **w**. The output is COVER – the indices of the sets selected and a vector $(y_i, i = 1, \ldots, m; v_j, j = 1, \ldots, n)$ that will later be proved to constitute a feasible dual solution.

THE DUAL-FEASIBLE MC ALGORITHM

Step 0: (initialize) $v_j = 0$, $j \in J$. $y_j = 0, i \in J$. COVER $= \emptyset$.

Step 1: Let $i \in I$. Let $w_k = \min\{w_j | j \in J\text{-COVER and } a_{ij} = 1\}$. ($k$ is the minimum cost column covering row i.) If no such minimum exists, stop - the problem is infeasible.

Step 2: Set $y_i \leftarrow y_i + w_k$. COVER $\leftarrow$ COVER$\cup\{k\}$. For all $j \in J$ such that $a_{ij} = 1$ set $w_j \leftarrow w_j - w_k$. If $w_j < 0$ then $v_j \leftarrow v_j - w_j$ and $w_j \leftarrow 0$.

Step 3: Set $b_i \leftarrow b_i - 1, i = 1, \ldots, m$. For all i' such that $b_{i'} = 0$, $I \leftarrow I - \{i'\}$. If $I = \emptyset$ stop. Else go to Step 1.

The algorithm repeats Step 1 at most n times, since if following n iterations the set I is not yet empty, then there is no feasible solution. This could occur for instance

if the amount of required coverage exceeds the number of covering sets, n. At each iteration there are at most $(\max\{n, m\})$ operations resulting in a total complexity of $O(\max\{n, m\} \cdot n)$.

The output of the algorithm is the set COVER of indices of the selected sets that multicover all elements, or a statement that the problem is infeasible. We shall now prove that dual vector derived is maximal and satisfies the three properties.

In the proofs of the facts that follow we shall use the notation $S_j = \{i \,|\, a_{ij} = 1\}$, i.e., S_j denotes the j^{th} set.

Fact 1. For each $j \in$ COVER, $w_j = \sum_{i \in S_j} y_i - v_j$.

Proof. By construction $w_j + v_j = \sum_{i \in S_j} y_i$. ■

Fact 2. $\sum_{i \in S_j} y_i \leq w_j + v_j \quad \forall j \in J$.

Proof. This follows from Fact 1 and from Step 1 since the minimum cost column is always selected. ■

Fact 3. The output of the algorithm $(\mathbf{y}, \mathbf{v}) = \left(\{y_i\}_{i=1}^m, \{v_j\}_{j=1}^n\right)$ is a feasible solution to the dual problem.

Proof. First, y_i, v_j are always nonnegative. This follows since y_i is equal to a cost w_j during one of the iterations and the $w_j's$ are always maintained as nonnegative numbers. Each v_j is a sum of positive numbers, and hence, nonnegative as well. Finally, Fact 2 establishes the feasibility with respect to the constraints. ■

Fact 4. $v_j = 0$ for all $j \in J$-COVER. This fact follows from the selection made at Step 1 of the algorithm.

The following lemma is useful in the proof of property (iii).

LEMMA 3.5 $\sum_{j \in \text{COVER}} v_j \leq \sum_{i \in I} (b_i - 1) y_i$ (note that $b_i \geq 1, i \in I$).

Proof. The values of the left-hand side and the right hand side of the inequality vary during the algorithm's iteration.

We let the value of v_j and y_i after iteration t be denoted by $v_j^{(t)}$ and $y_i^{(t)}$ respectively. Let T be the number of iterations. We shall prove by induction on t that

$$\sum_{j \in \text{COVER}} v_j^{(t)} \leq \sum_{i \in I} (b_i - 1) y_i^{(t)}, \quad i = 1, \ldots, T.$$

For $t = 1$, the left-hand side is zero and the right-hand side nonnegative. We shall assume by induction that the inequality holds for $t = 1, \ldots, l-1$ and prove for l.

Let $M = w_k$ be the minimum column cost selected at iteration l; then the right-hand side increases by $(b_i - 1) \cdot M$ with y_i increasing by M. Each $v_j^{(l)}$ might be increased by at most M compared to the previous iteration but for no more than $(b_i - 1)$ columns. This is the case since a cost of a column could become negative (thus triggering the increase in $v_j^{(l)}$), only if it is already b_i columns or more covering row i in COVER. Then this row

would have been removed from the set I, and thus could not be considered at iteration l. Therefore, the inequality is preserved at each iteration, and hence, the desired result. ■

From the proof of the theorem it follows that the heuristic solution value does not in fact exceed $(\max_{i \in I} \sum_{i \in \text{COVER}} a_{ij})$ times the value of the optimum. This quantity could be much smaller than p.

The derivation of the dual vector as a by-product of the heuristic also provides a certificate of optimality for the selected set COVER (or any other solution) satisfying $\sum_{i \in \text{COVER}} w_j = \sum_{i=1}^{n} b_i y_i - \sum_{j=1}^{n} v_j$.

THE OPTIMAL DUAL APPROACH FOR THE VERTEX COVER AND INDEPENDENT SET PROBLEMS: PREPROCESSING

3.7

As we saw earlier, finding the optimal dual solution rather than a feasible one does not offer improved approximation bounds in general but requires more running time. Still, for the vertex cover and independent set (and the more general integer programs with two variables per inequality), the optimal dual solution provides important information about the problem, and allows it to improve the approximation bounds. For vertex cover, the use of the optimal dual solution guarantees that any heuristic used along with that information as preprocessing, yields an approximation ratio strictly better than 2.

Many intuitively reasonable heuristics for the vertex cover problem, (with the exception of the LP-algorithm), may fare quite badly compared to the optimal solution to the problem. For instance, a natural heuristic to consider is to take the largest degree vertex in the graph for the unweighted problem. This is the greedy algorithm, and, as shown by Johnson [Joh74], this heuristic applied to a graph of maximum degree k may deliver a cover whose weight exceeds the weight of an optimal cover by a factor of $\mathcal{H}(k)$ even if all weights are unit.

Another heuristic is available when the set of vertices of the graph V is split into independent sets $\{V_1, \ldots, V_k\}$. (Methods of obtaining such a split will be described later.) Each set $V \setminus V_i$ is a cover. In particular, $V \setminus V_i$ of the smallest weight may seem to be a good candidate for a cover C. Still, the ratio $\frac{w(C)}{w(C^*)}$ may be arbitrarily large even when G is fixed. Indeed, consider the path with vertices $\{1, 2, 3, 4\}$ and weights $w_1 = w_4 = M$, $w_2 = w_3 = 1$ for some large M. When V is partitioned into two independent sets, the strategy proposed here yields a cover C with $w(C) = M + 1$ and yet the optimal cover C^* has $w(C^*) = 2$. Some more illustrations of such undesirable behavior of other heuristics are quite common. Still, it seems that the LP-heuristic alone employs relatively little information about the underlying structure of the graph, and one should be able to fare better with additional graph information taken into account.

This idea has motivated the use of the preprocessing procedure of [Hoc83] that makes use of the linear programming information *and* additional information about the graph. Indeed, the quality of the solution delivered by any heuristic can be improved if we first partition the graph into two subgraphs with the property that in one subgraph an optimal selection of a cover is known and in the other the weight of the optimal cover is at least half of the total weight of all vertices. The existence of such a partition, implied by the fractional solution to the problem, has been established by Nemhauser and Trotter [NT75].

More precisely, Nemhauser and Trotter [NT75], Balinski and Spielberg [BS69], and Lorentzen [Lor66] have shown that there exists an optimal solution to the linear programming relaxation of (VC), $\mathbf{x}^*$, such that $x_j^* \in \{0, 1, \frac{1}{2}\}$. We call this property the *half integrality property*. Nemhauser and Trotter have further proved that there exists an optimal integer solution that is equal to $\mathbf{x}^*$ in its integer components. We refer to this property as the *"fixing variables" property*. Both the half integrality property and the "fixing variables" property were proved to hold also for IP2 (see [HMNT93] or 3.8).

One possible algorithmic use of fixing variables is that an optimal integer solution may be obtained by rounding the components of $\mathbf{x}^*$ that are equal to $\frac{1}{2}$. The rounding could be up or down to 1 or 0 respectively. Since there are 2^n possible rounding schemes (some of which may not lead to a feasible solution), this fact in itself does not aid in speeding up the search for an optimal solution. It does, however, provide us with a head-start in the search towards a solution: consider an optimal solution $\mathbf{x}^*$ to (VCR), and the implied partition:

$$j \in P \text{ if } x_j = 1$$

$$j \in Q \text{ if } x_j = \frac{1}{2}$$

$$j \in R \text{ if } x_j = 0.$$

Then, using the fixing variables property we conclude,

(i) at least one optimal cover in C contains P,

(ii) each vertex in R has all its neighbors in P,

(iii) each cover in G has weight at least $w(P) + \frac{1}{2}w(Q)$.

From (i) and (ii), it follows instantly that at least one optimal cover in G consists of the set P and of an optimal cover in the subgraph H induced by Q. Thus, it suffices to find an optimal cover in H; working with H rather than with G is what we mean by "fixing variables."

Fixing variables is a trick which can be applied not only in the context of finding optimal covers but also in the context of heuristics for finding near-optimal covers. In this context, the trick has a nice corollary: If C is any cover in H, then (by (ii)) $P \cup C$ is a cover in G, and (by (iii)) its weight is at most twice the weight of an optimal cover. Thus any heuristic for finding near-optimal covers can be made to deliver a cover whose weight is at most twice the weight of an optimal cover: it suffices to preprocess G by finding P, Q, R, and then to apply the heuristic to H rather than directly to G. Formally,

let C^H be the cover delivered by the heuristic on the subgraph H. Then,

$$\frac{w(C^H \cup P)}{w(C^*)} \leq \frac{w(C^H) + w(P)}{\frac{1}{2}w(Q) + w(P)} \leq 2\frac{w(C^H)}{w(Q)}.$$

The consequence of using the preprocessing technique is that *any* heuristic for the vertex cover problem delivers a solution that is less than twice times the optimum. C^H is a subset of Q, and one can always remove just one vertex from Q that is of maximum weight and consider the rest as the heuristic cover.

The preprocessing technique has become the basis of most "good" heuristics for the vertex cover problem, and it can be used to improve by a factor of two the approximation ratio for the independent set problems. Unlike the vertex cover, we cannot guarantee an approximation ratio better than half (or factor of two off the optimum) for the independent set problem. To see why, consider a heuristic applied to the subgraph H delivering an independent set IS^H. The ratio of the weight of the resulting independent set to the optimal independent set IS^* is,

$$\frac{w(IS^H \cup R)}{w(IS^*)} \geq \frac{w(IS^H) + w(R)}{\frac{1}{2}w(Q) + w(R)} \geq 2\frac{w(IS^H)}{w(Q)}.$$

Now IS^H could be arbitrarily small, so the ratio may be arbitrarily close to 0. If on the other hand the heuristic procedure guarantees a certain positive ratio, the use of the preprocessing technique may double this ratio. Whether or not the ratio is doubled depends on whether the graph property is hereditary and maintained for the subgraph. When applying the procedure to a graph with a small largest claw number, for instance, there is no improvement with preprocessing (as this property is not hereditary), and the bound on the optimum relies on the bounded claw number of the graph. This algorithm will be demonstrated in a later section.

Pulleyblank [Pul79] observed the following with regard to the size of the subsets P and Q. He proved that almost all graphs (randomly generated) have an LP-relaxation for which the solution is a vector of $\frac{1}{2}$s and no integer entries. In that sense, the use of the preprocessing is more to guarantee that the graph has no integer nodes, rather than to attempt to find many integer values.

3.7.1 THE COMPLEXITY OF THE LP-RELAXATION OF VERTEX COVER AND INDEPENDENT SET

In order to apply the preprocessing solution the optimal LP solution must be available. Although in principle it is possible to solve linear programs in polynomial time (using Ellipsoid method or interior-point methods), such procedures are less efficient than many combinatorial algorithms. In particular, the LP-relaxation of the vertex cover is solvable by the max-flow min-cut algorithm.

The LP-relaxation of the vertex cover problem can be solved by finding an optimal cover in a bipartite graph with two vertices for each vertex in the original graph, and two edges for each edge in the original graph (see Figure 3.3). In the bipartite graph a vertex cover may be identified from the solution of a corresponding minimum cut problem. Specifically, as suggested by Edmonds and Pulleyblank and noted in [NT75], the LP-

relaxation can be solved by finding an optimal cover C in the bipartite graph with two vertices a_j, b_j of weight w_j for each vertex j of G, and two edges (a_i, b_j), (a_j, b_i) for each edge (i, j) of G: then it suffices to set

$$x_j = 1 \text{ if } a_j, b_j \in C,$$

$$x_j = \frac{1}{2} \text{ if } a_j \in C, b_j \notin C, \text{ or } a_j \notin C, b_j \in C,$$

$$x_j = 0 \text{ if } a_j \notin C, b_j \notin C.$$

In turn, the problem of finding C can be reduced into a minimum cut problem: the bipartite graph can be converted into a network by making each edge (a_i, b_j) into a directed arc (a_i, b_j) of an infinite capacity, adding a source s with an arc (s, a_i) of capacity w_i for each i, and adding a sink t with an arc (b_j, t) of capacity w_j for each j. Now a minimum cut (S, T) with $s \in S$ and $t \in T$ points out the desired C: it suffices to set $a_i \in C$ iff $a_i \in T$ and $b_j \in C$ iff $b_j \in S$. A description of the resulting graph is given in Figure 3.3.

The minimum cut can be found by efficient algorithms for maximum flow on (bipartite) graphs. For instance, if one uses Goldberg and Tarjan's algorithm, [GT88], it takes only $O(mn \log \frac{n^2}{m})$ steps to preprocess G with n vertices and m edges by partitioning the set of vertices into P, Q and R. This algorithm is a special case of the algorithm used to find the half integer solution for IP2 where min cut is also used for preprocessing (see Section 3.8).

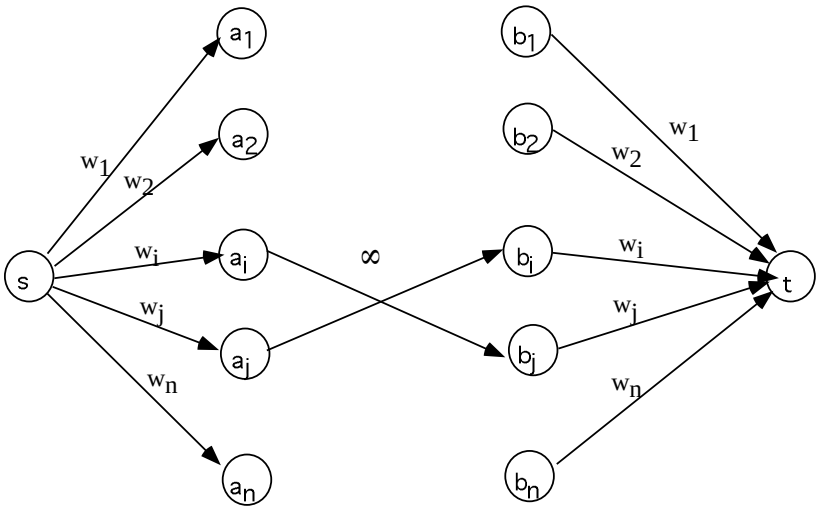

FIGURE 3.3

We now present an alternative method of reducing the LP-relaxation to a flow problem that is illuminating as to why min cut solves this problem. Consider the LP-relaxation of the vertex cover problem (VCR),

$$\text{(VCR)} \qquad \begin{array}{ll} \text{Min} & \sum_{j=1}^{n} w_j x_j \\ \text{subject to} & x_i + x_j \geq 1 \quad \text{(for every edge } (i,j) \text{ in the graph)} \\ & 0 \leq x_j \leq 1 \quad (j = 1, \ldots, n). \end{array}$$

Replace each variable x_j by two variables, x_j^+ and x_j^-, and each inequality by two inequalities:

$$\begin{aligned} x_i^+ - x_j^- &\geq 1 \\ -x_i^- + x_j^+ &\geq 1 \,. \end{aligned}$$

The two inequalities have one 1 and one -1 in each, and thus correspond to a dual of a network flow problem. The upper and lower bounds constraints are transformed to

$$\begin{aligned} 0 \leq x_j^+ &\leq 1 \\ -1 \leq x_j^- &\leq 0 \,. \end{aligned}$$

In the objective function, the variable x_j is substituted by $\frac{1}{2}(x_j^+ - x_j^-)$.

The resulting constraint matrix of the new problem is totally unimodular. Hence, the linear programming (optimal basic) solution is integer, and in particular can be obtained using a minimum cut algorithm. When the original variables are recovered, they are integer multiples of $\frac{1}{2}$.

We will see in Section 3.8 that this transformation is applicable to all integer programs with two variables per inequality. That, in addition to the "fixing variables" property that applies, guarantees that *any* heuristic will give a bound of 2 or better.

Remark: When the problem is *unweighted*, the network flow that solves the LP relaxation is defined on *simple* networks. These are networks with all arcs of capacity 1, and every node has either one incoming arc or one outgoing arc. In this case, the arcs in the bipartition of the type (a_i, b_j) can be assigned capacity 1 instead of ∞. For simple networks, Dinic's algorithm for maximum flow works in $O(\sqrt{n}m)$ time—a significant improvement in running time.

3.7.2 EASILY COLORABLE GRAPHS

It is now demonstrated how to exploit certain graph properties with the preprocessing technique so as to obtain approximation factors better than 2 for the vertex cover problem, and improved approximations for the independent set problem. For many classes of graphs it is easy to split the nodes into independent sets. Assigning each independent set a color results in a valid coloring with adjacent vertices having distinct colors.

THEOREM 3.2 Let G be a weighted graph with n vertices and m edges; let k be an integer greater than one. If it takes only s steps to color the vertices of G in k colors, then it takes only $s + O(nm \log \frac{n^2}{m})$ steps to find an independent set whose weight is at least $2/k$ times the weight of an optimal independent set and to find a cover whose weight is at most $2 - 2/k$ times the weight of an optimal cover.

Proof. It takes only $s + O(nm \log \frac{n^2}{m})$ steps to color G in k colors and to find the set P, Q, R of the preceding section. (Note that it suffices to color only the vertices of Q.) The coloring of G splits Q into k color classes; if S denotes the heaviest of them then $W(S) \geq W(Q)/k$. The set $R \cup S$ is independent since

$$w(R \cup S) \geq w(R) + \frac{1}{k} w(Q) \geq \frac{2}{k} \left((w(R) + \frac{1}{2} w(Q) \right),$$

its weight is at least $2/k$ times the weight of an optimal independent set. The complement C of $R \cup S$ is a cover since

$$w(C) \leq w(P) + \frac{k-1}{k} w(Q) \leq \frac{2(k-1)}{k} \left(w(P) + \frac{1}{2} w(Q) \right),$$

its weight is at most $2 - 2/k$ times the weight of an optimal cover. ■

The remainder of this section consists of various corollaries of Theorem 3.2. To begin with, let $D(G)$ denote the largest d such that G contains a subgraph in which each vertex has degree at least d. As proved by Szekeres and Wilf [SW68], every graph G can be colored in $D(G)+1$ colors. For the sake of completeness, we shall describe a way of finding such a coloring and evaluating $D(G)$ in only $O(n+m)$ steps. To evaluate $D(G)$, it suffices to dismantle G by successive removals of vertices of minimum degree.

MAXIMUM MINIMUM DEGREE SUBGRAPH

Step 0: Set $d = 0$.

Step 1: If G has no vertices left, then stop; otherwise choose a vertex v of the smallest degree.

Step 2: Replace d by the maximum of d and the degree of v. Then remove v (and all the edges incident with v) from G and return to Step 1.

If v_i denotes the vertex removed from G in the ith iteration, then each v_i has at most d neighbors among the vertices $v_{i+l}, v_{i+2}, \ldots, v_n$. To color G in no more than $d+1$ colors, it suffices to scan the sequence of $v_i's$ from v_n to v_l, assigning to each v_i the smallest positive integer not yet assigned to any of its neighbors.

COROLLARY 3.2 It takes only $O(nm \log \frac{n^2}{m})$ steps to find, in any weighted graph G with n vertices and m edges such that $m > 0$, an independent set whose weight is at least $2/(D(G)+1)$ times the weight of an optimal independent set and a cover whose weight is at most $2 - 2/(D(G)+1)$ times the weight of an optimal cover.

The celebrated theorem of Brooks [Bro41] asserts the following: if G is a connected graph of a maximum degree Δ such that $\Delta > 3$ and if G is not the complete graph with $\Delta+1$ vertices, then G is Δ-colorable. An elegant and constructive proof of this theorem, due to Lovász [Lov75a], provides an algorithm which finds the coloring in only $O(\Delta n)$ steps. (The algorithm requires finding cutpoints and endblocks in a graph. This can be done in $O(m)$ steps by depth-first search as described, for instance, in [Baa78]).

COROLLARY 3.3 It takes only $O(\Delta n^2 \log \frac{n^2}{m})$ steps to find, in any weighted graph with n vertices and a maximum degree Δ such that $\Delta \geq 2$, an independent set whose weight is at least $2/\Delta$ times the weight of an optimal independent set and a cover whose weight is at most $2 - 2/\Delta$ times the weight of an optimal cover.

Proof. We may assume that $\Delta \geq 3$; otherwise each component is a cycle or a path and a straightforward dynamic programming algorithm finds an optimal independent set and an optimal cover in only $O(n)$ steps. Furthermore, we may assume that the graph is connected; otherwise each component may be treated separately. Finally, we may assume that the graph is not complete: otherwise an optimal independent set and an optimal cover may be found trivially in $O(n)$ steps. But then the desired conclusion follows directly from Brooks' theorem and Theorem 3.2. ■

We will show in the next subsection that the $\frac{2}{\Delta}$ guarantee for independent set can be improved by using a better partition.

The coloration heuristics are not counterexamples to our conjecture that vertex cover is impossible to approximate within a ratio strictly less than 2. To show that, we let a graph G be defined as follows. Consider Δ Δ-cliques and Δ $(\Delta - 1)$-independent sets. Each clique has one edge connecting it to one of the vertices of an independent set. $\Delta - 1$ of the independent sets are one set of vertices in a complete bipartite graph with the Δth independent set as the second set of vertices. For such family of graphs, one can easily verify that G is Δ chromatic and $G = H$. One feasible[1] Δ-coloration consists of each one of the independent sets colored by one of the Δ-colors. The heuristic then delivers a cover C of size $(2\Delta - 1)(\Delta - 1)$. The optimum cover C^* is of size $\Delta(\Delta - 1) + (\Delta - 1)$ and the ratio

$$w(C)/w(C^*) = 2 - \frac{2 - 3/\Delta}{\Delta - 1/\Delta}$$

which could be arbitrarily close to 2.

The standard proof due to Heawood that every planar graph is five-colorable (see, for instance, [Har69]) has been converted into linear time algorithms [CNS81] [MST80].

COROLLARY 3.4 It takes only $O(n^2 \log \frac{n^2}{m})$ steps to find, in any weighted planar graph with n vertices, an independent set whose weight is at least 0.4 times the weight of an optimal independent set and a cover whose weight is at most 1.6 times the weight of an optimal cover.

Furthermore, the proof that every planar graph is four-colorable [AH77] [AHK77] is convertible into an algorithm which actually finds the coloring in a polynomial number of steps.

COROLLARY 3.5 It takes a polynomial number of steps to find, in any weighted planar graph, an independent set whose weight is at least 0.5 times the weight of an

[1] In order to make this coloration unique we add a few edges to the graph: The vertices in the cliques that connect each clique to each independent set are linked together to make a complete subgraph. The i^{th} independent set is connected to all these vertices except for the i^{th} vertex.

optimal independent set and a cover whose weight is at most 1.5 times the weight of an optimal cover.

3.7.3 A GREEDY ALGORITHM FOR INDEPENDENT SET IN UNWEIGHTED GRAPHS

Sparse graphs have large independent sets. More precisely, the celebrated Theorem of Turán [Tur41] asserts that every graph with n vertices and an average degree δ (this quantity is not necessarily an integer) contains a independent set of size at least $\frac{n}{\delta+1}$. An elegant proof of Turán's theorem, due to Erdös [Erd70], is easily converted into the following algorithm for finding a independent set S in at most $O(m)$ steps.

GREEDY ALGORITHM [ERDÖS]

Step 0: Set $S = 0$.

Step 1: If G has no vertices then stop; otherwise choose a vertex v with the smallest degree d in the current graph.

Step 2: Add v to S, delete v and all its neighbors (along with all the edges incident with at least one of these vertices) from G, and return to Step 1.

To show that the size of the independent set S delivered upon termination is at least $\frac{n}{\delta+1}$, we observe that whenever a vertex v_i of degree d_i (this is the degree of v_i in the reduced graph from which v_i is selected and subsequently removed) is chosen and deleted, we eliminate a total of $d_i + 1$ vertices from the graph and the sum of the degrees of the vertices deleted is at least $d_i(d_i + 1)$. If $q = |S|$ is the number of vertex selections performed in the greedy algorithm, then

$$\sum_{i=1}^{q} d_i(d_i+1) \le n\delta \quad \text{and} \quad \sum_{i=1}^{q}(d_i+1) = n \ . \tag{3.8}$$

By adding these two equations together and then applying the Cauchy-Schwarz inequality, we get that

$$n(\delta+1) \ge \sum_{i=1}^{q}(d_i+1)^2 \ge \frac{n^2}{q} \ ,$$

from which it follows that $q \ge \frac{n}{\delta+1}$.

By using the greedy algorithm in conjunction with preprocessing, we get the following result.

THEOREM 3.3 [Hoc83] In any graph G with n vertices and average degree δ it takes $O(\delta n^{\frac{3}{2}})$ steps to find an independent set of size at least $\frac{2}{\delta+1}$ times the size of maximum independent set.

Proof. Preprocessing an unweighted graph can be executed in only $O(m\sqrt{n})$ steps [HK73]. Once the partition P, Q, R is obtained we apply the algorithm above to the subgraph H. The total number of steps does not exceed $O(\delta n^{\frac{3}{2}})$. The size of the independent set delivered by the algorithm is at least $|R| + \frac{|Q|}{\delta_H+1}$, where δ_H is the average degree in H. (Incidentally, note that $\delta_H \geq 2$ as there is always a solution with no vertices of degree one in the subgraph H.) Now we note that:

Fact 1. G is a connected graph, hence,

$$\delta \geq \frac{|Q|\delta_H + |R| + |P|}{|Q| + |R| + |P|}$$

(note that $n = |Q| + |R| + |P|$).

Fact 2. $|R| \geq |P|$, otherwise setting $G = H$ (i.e., all vertices are assigned the value $\frac{1}{2}$) implies an "LP relaxation" solution of value larger than $|R| + \frac{1}{2}|Q|$, contradiction.

To complete the proof it suffices to show (using Fact 1) that

$$\frac{|R| + \frac{|Q|}{\delta_H+1}}{|R| + \frac{1}{2}|Q|} \geq 2\left(\frac{|Q|\delta_H + |R| + |P|}{|Q| + |R| + |P|}\right)^{-1}.$$

Rearranging this inequality we reduce it to

$$|Q|\,(|R|\delta_H(\delta_H - 1) - |P|(\delta_H - 1) - |P|(\delta_H - 1)) \geq 0.$$

The validity of this inequality follows easily from Fact 2. ■

Halldórsson and Radhakrishnan ([HR94]) have recently tightened this analysis to achieve an improved bound on q. Consider an optimal independent set, and let k_i be the number of nodes from this independent set deleted at stage i of the greedy algorithm. Then, because an edge can have only one of its endpoints in the maximum independent set, the equations (3.8) can be tightened:

$$\sum_{i=1}^{q} d_i(d_i + 1) + k_i(k_i - 1) \leq n\delta \text{ and } \sum_{i=1}^{q}(d_i + 1) = n\ .$$

Adding these two equations along with $\sum_{i=1}^{q} k_i = IS^*$, and applying the Cauchy-Schwarz inequality yields,

$$(\delta + 1)n + \alpha \geq \sum_{i=1}^{q}(d_i + 1)^2 + k_i^2 \geq \frac{n^2 + (IS^*)^2}{q}\ ,$$

which implies that $q \geq \frac{n^2+(IS^*)^2}{n(\delta+1)+IS^*}$. Because this quantity is maximized at $IS^* = n$, the following improved bound for the quality of the greedy solution is found:

$$q \geq \frac{2}{\delta + 2}\ .$$

When used in conjunction with the preprocessing technique, as in Theorem 3.3, a better performance bound of $\frac{5}{2\delta+3}$ is achieved.

[HR94] also provides an analysis that yields a performance guarantee in terms of the maximum degree Δ in an unweighted graph of $q \geq \frac{3}{\Delta+2}$ for the greedy algorithm. This is better than the bound in Corollary 3.3 when the graph is unweighted and $\Delta \geq 5$.

As we discuss in the next subsection, the idea of *subgraph removal* can be incorporated to further improve the performance guarantee given for the greedy algorithm.

3.7.4 A LOCAL-RATIO THEOREM AND SUBGRAPH REMOVAL

In some cases, the absence of a particular family of subgraphs $\mathcal{H}$ (for example, odd cycles) from a graph G can imply an improved approximation guarantee for finding an optimal structure in G. Bar-Yehuda and Even [BE85] have developed a local-ratio theorem that, by removing problematic subgraphs, yields several new approximation algorithms which improve upon previously known results for the weighted vertex cover problem. Their main result is a $(2 - \frac{\log\log n}{2\log n})$-approximation algorithm that relies on eliminating "small" odd cycles. This subsection describes their approach and its extension to other approximation problems.

Let $\mathcal{H}$ be a set of graphs. Let $A_{\mathcal{H}}$ be an approximation algorithm for the weighted vertex cover problem. Consider the following algorithm to find a vertex cover in the graph $G = (V, E)$, with weight function w.

SUBGRAPH REMOVAL ALGORITHM [BAR-YEHUDA AND EVEN]

Step 0: Set $w_0 \leftarrow w$.

Step 1: While there exists a subgraph H of G that is isomorphic to some member of $\mathcal{H}$ and whose vertices have positive weight, do: $\forall v \in V(H)$ set $w_0(v) \leftarrow w_0(v) - \delta$, where $\delta \leftarrow \min\{w_0(v) | v \in V(H)\}$.

Step 2: Set: $VC_0 \leftarrow \{v \in G | w_0(v) = 0\}$; $V_1 \leftarrow V - VC_0$.

Step 3: Let VC_1 be returned by applying $A_{\mathcal{H}}$ on $G(V_1)$, with the weight function w_0. Return $VC \leftarrow VC_0 \cup VC_1$.

For each entry in $H \in \mathcal{H}$, let $r_H = \frac{n_H}{c_H}$, where n_H is the number of vertices in H and c_H is the cardinality of a minimum unweighted vertex cover in H. r_H is called the *local-ratio* of the graph H. The quality of solution returned by the above algorithm is given by the following theorem from [BE85]:

THEOREM 3.4 Local-Ratio Theorem Let $r_{\mathcal{H}} = \max_{H \in \mathcal{H}}\{r_H\}$ and $r_{A_{\mathcal{H}}}$ be the approximation factor guaranteed by the algorithm $A_{\mathcal{H}}$ on input $G(V_1)$. Then, the vertex cover VC returned by the Subgraph Removal Algorithm has weight at most $r = \max\{r_{\mathcal{H}}, r_{A_{\mathcal{H}}}\}$ times that of the optimal vertex cover.

In order to establish Theorem 3.4, we need a preliminary lemma.

LEMMA 3.6 Let $G = (V, E)$ be a graph, and w, w_1, and w_2 be weight functions on the vertices V, with optimal vertex covers VC^*, VC_1^* and VC_2^*, respectively. Suppose that $w(v) \geq w_1(v) + w_2(v)$, for every $v \in V$. Then,

$$w(VC^*) \geq w_1(VC_1^*) + w_2(VC_2^*) \ .$$

Proof.

$$w(VC^*) = \sum_{v \in VC^*} w(v) \geq \sum_{v \in VC^*} (w_1(v) + w_2(v))$$
$$= w_1(VC^*) + w_2(VC^*) \geq w_1(VC_1^*) + w_2(VC_2^*) \ .$$

■

Proof. (**Local-Ratio Theorem**) The proof is by induction on k, the number of times that the *do-while* loop in Step 1 of the Subgraph Removal Algorithm is iterated through. For $k = 0$ the theorem is obviously true.

Now consider the case $k = 1$. Let VC^* and VC_0^* be the optimal solutions with respect to w and w_0, and let VC be the vertex cover returned by the algorithm. Then,

$$w(VC) \leq w_0(VC) + \delta n_H \leq r_{A_{\mathcal{H}}} w_0(VC^*) + r_H \delta c_H$$
$$\leq r\,(w_0(VC^*) + \delta c_H) \leq r\,w(VC^*) \ .$$

For $k > 1$, imagine running Step 1 through one iteration. Then, by considering the remainder of the algorithm as "$A_{\mathcal{H}}$" with performance guarantee r (from the induction hypothesis), we are in the $k = 1$ case, from which the result follows. ■

By selecting $\mathcal{H}$ appropriately and designing the approximation algorithm $A_{\mathcal{H}}$ to take advantage of the absence of such subgraphs, the *Subgraph Removal Algorithm* achieves approximation procedures with improved efficiency and/or approximation guarantee over previously known algorithms. The following is a summary of the results presented in [BE85].

1. $\mathcal{H}$ *is an edge:* An edge has a local-ratio of 2. By removing edges during Step 1 of the *Subgraph Removal Algorithm*, we are left with an empty graph at Step 2 (for which $r_{\mathcal{H}}$ vacuously equals 1). Thus, the algorithm gives a 2-approximation.
2. $\mathcal{H}$ *is a triangle:* By removing triangles during Step 1, which have a local-ratio of 1.5, we are left with a triangle-free graph at Step 2. Then, by using the $2 - \frac{2}{k}$ approximation algorithm from [Hoc83] as $A_{\mathcal{H}}$, where k is the number of colors needed to color the remaining graph, two results follow:
   a. Wigderson has shown [Wig83] that triangle free graphs can be colored with $k = 2\sqrt{n}$ colors in linear time. This yields a $\min\{1.5, 2 - \frac{1}{\sqrt{n}}\} = 2 - \frac{1}{\sqrt{n}}$ (for $n \geq 4$) approximation guarantee for general graphs. (Halldórsson [H94] has noted the existence of a nice algorithm [She83] that colors triangle free graphs with $k = 2\sqrt{\frac{n}{\log n}}$ colors, yielding a $2 - \sqrt{\frac{\log n}{n}}$ approximation guarantee.)
   b. Triangle-free planar graphs can be colored with $k = 4$ colors in linear time (see [Har69]), yielding an algorithm that matches the known 1.5 approximation guarantee of [Hoc83] for planar graphs. The advantage is that the complexity of the general 4-coloring algorithms for planar graphs is avoided.
3. $\mathcal{H}$ *is the set of "small" odd cycles:* Odd cycles up to length $2k - 1$, where $(2k-1)^k \geq n$ (so, $k \leq \frac{2\log n}{\log\log n}$), are removed. This set of cycles has local ratio $r_{\mathcal{H}} = \frac{2k-1}{k} = 2 - \frac{1}{k}$. For $A_{\mathcal{H}}$ we use an algorithm, with the same performance ratio, developed in [BE85] for graphs in which all odd cycles have length at

least $2k+3$. This yields an approximation ratio guarantee of $2-\frac{1}{k} \leq 2-\frac{\log\log n}{2\log n}$. (Monien and Speckenmeyer [MS85] have used a similar approach to achieve improved results for the unweighted vertex cover problem.)

Halldórsson and Radhakrishnan [HR94] have used the strategy of triangle removal, in conjunction with preprocessing, to achieve several additional results for finding vertex covers in unweighted graphs:

1. Removing triangles and applying the coloring algorithm of [She83] yields a $2-\frac{\log\Delta+O(1)}{\Delta}$ approximation guarantee, where Δ is the maximum degree of a vertex in the graph.
2. Consider a graph that is p-claw free. After removing triangles, the size of the largest claw is equal to the maximum degree of the remaining graph. Thus, the preceding result holds with Δ replaced by $p-1$.

Halldórsson and Radhakrishnan [HR94] also employ a strategy of subgraph removal to achieve improved approximation algorithms for finding large (unweighted) independent sets. Their general schema, which entails removing all cliques of a given size from the graph and then running any particular independent set algorithm on the resulting graph, is used to achieve improved performance bounds for the particular independent set algorithm. Used in conjunction with the greedy algorithm, for instance, they achieve an asymptotic performance ratio of $\frac{3.76}{\Delta}$ by removing 8-cliques. Better performance bounds may be achievable if it is used in conjunction with other heuristics.

3.7.5 ADDITIONAL ALGORITHMS WITHOUT PREPROCESSING

3.7.5.1 Independent Set in Weighted Graphs

By applying a graph theoretic result of Lóvasz [Lov66], it is possible to improve the approximation guarantee for the weighted independent set problem in bounded degree graphs from $\frac{2}{\Delta}$ to $\frac{1}{\lceil\frac{\Delta+1}{3}\rceil}$, where Δ is the maximum degree of a vertex in the graph.

The idea is to consider a partition of a graph into k subgraphs, for some integer k; that is, partition the vertices into k subsets and consider the k subgraphs induced by each of the k subsets. Being able to find an optimal independent set in each subgraph implies being able to find an independent set with weight within a factor of $\frac{1}{k}$ of the optimal in the original graph. This fact is established in the next theorem.

THEOREM 3.5 Consider a weighted graph $G=(V,E)$, and let $V_1,\ldots,V_k$ be a partition of the vertices V into k subsets. If IS_i^* is an optimal independent set in G_i (where G_i is the subgraph of G induced by the vertices V_i) for $i=1,\ldots,k$, and IS^* an optimal independent set in G, then

$$\max_{i=1,\ldots,k}\{w(IS_i^*)\} \geq \frac{1}{k}w(IS^*) .$$

Proof. Because IS^* is an independent set in G, $IS^* \cup V_i$ is an independent set in G_i. So,

$$\sum_{i=1}^{k} w(IS_i^*) \geq \sum_{i=1}^{k} (w(IS^*) \cap V_i) = w(IS^*) .$$

Now, the result follows by the pigeonhole principle. ■

Thus, being able to partition a graph into k subgraphs in which optimal independent sets can be found in polynomial time leads to a $\frac{1}{k}$-approximation algorithm for weighted independent set. Of the optimal independent sets in the subgraphs, select the one with maximum weight. More generally, being able to solve the independent set problem for each subgraph within a factor β of the optimal implies a $\frac{\beta}{k}$-approximation algorithm. Halldórson has noted the existence of a partitioning that can be used in this manner by applying the following theorem due to Lovász:

THEOREM 3.6 Lovász Let $G(V, E)$ be a graph with maximum degree Δ. Let k be any integer such that $1 \leq k \leq \Delta$, and let $\Delta_1, \ldots, \Delta_k$ be nonnegative integers such that,

$$\Delta_1 + \Delta_2 + \ldots + \Delta_k = \Delta - k + 1 .$$

Then, V can be partitioned into k subsets $V_1, \ldots, V_k$, such that Δ_i is the maximum degree of a vertex in the subgraph of G induced by V_i, for $i = 1, \ldots, k$.

Moreover, a crude analysis of the algorithm implied by the proof of the above theorem shows that the partitioning of the graph can be carried out in $O(mk)$ time. The previous two theorems lead to the following corollary:

COROLLARY 3.6 Let G be a weighted graph with maximum degree Δ. An independent set with weight within a factor of $\frac{1}{\lceil \frac{\Delta+1}{3} \rceil}$ of the optimal can be found in $O(m\Delta)$ time.

Proof. Applying Theorem 3.6, a graph can be partitioned into $k = \lceil \frac{\Delta+1}{3} \rceil$ subgraphs, each with maximum degree 2, in $O(m\Delta)$ time. Now, an optimal independent set in such graphs can be found in linear time. Thus, by Theorem 3.5, we can find an independent set with weight within $\frac{1}{\lceil \frac{\Delta+1}{3} \rceil}$ of the optimal. ■

3.7.5.2 Vertex cover in unweighted large min-degree graphs

Karpinski and Zelikovsky [KZ95] recently proposed an algorithm for which the approximation is improved for graphs with large minimum degree. Let a graph be called $\underline{\delta}$-dense if each vertex is adjacent to at least $\underline{\delta} n$ vertices. Let $N(v)$ be the set of neighbors of v in $G = (V, E)$. The following is a $\frac{2}{1+\underline{\delta}}$- approximation algorithm:

procedure $\frac{2}{1+\underline{\delta}}$- approximation
for all $v \in V$ do
    $V(v) \leftarrow V \setminus \{N(v) \cup v\}$
    apply **dual-feasible I algorithm** to find a feasible vertex cover
    $VC(v)$ in the graph induced on $V(v)$.
    $VC(v) \leftarrow VC(v) \cup \{N(v)\}$
Return $VC(u)$ where $|VC(u)| = \min_{v \in V} |VC(v)|$.

The algorithm makes n calls to dual-feasible I, hence its complexity is $O(mn)$. The proof that the bound is valid is derived by assessing the ratios for the cases when the optimum cover OPT satisfies $|OPT| \leq (1-\underline{\delta})n$ and $|OPT| \geq (1-\underline{\delta})n$.

3.7.6 SUMMARY OF APPROXIMATIONS FOR VERTEX COVER AND INDEPENDENT SET

We present here two tables along with a notation legend with the best known approximation results to date for vertex cover and independent set. The following legend explains the meanings of the symbols used in Table 3.1:

Symbol	*Meaning*
n	number of vertices in the graph
m	number of edges in the graph
α	value of optimal independent set
χ	chromatic number
Δ	maximum vertex degree
δ	average vertex degree
$D(G)$	$\max_{H \subseteq G}\{\min_{v \in H}\{\text{degree}(v)\}\}$
p-claw	a subset of $(p+1)$ vertices that induces a p-star
$T(n,m)$	complexity of finding a minimum cut in a network with n nodes, m arcs
S	complexity of applying Shearer's coloring algorithm
$\underline{\delta}$	minimum vertex degree

References and comments: (1) [Hoc83]; (2) [Hoc83], Δ-coloring via Brooks' theorem; (3) [HR94]; (4) [HR94]; (5) [Hoc83],(δ_H is ave degree in subgraph H.); (6) [Hoc83]; (7) [Hoc83]; this running time is from [BE85]; (8) [Bak83], approximation scheme; (9) [BE85], via coloring algorithm in [Wig83]; (10) [BE85]; (11) [Hoc83] c is a fixed constant; (12) [MS85]; (13) [Hoc83]; (14) [YG92]; (15) [HR94]; (16) [KZ95]; (17) [Hoc83]; (18) [H94], using graph decomposition of [Lov66]; (19) [HR94], analysis of greedy algorithm; (20) [Hoc83]; (21) [HR94]; (22) [HR94], analysis of greedy algorithm; (23) [Hoc83]; (24) [Hoc83]; (25) [Bak83], approximation scheme; (26) [Hoc83]; (27) [YG92].

Vertex Cover Problem					
Graph Parameter	*Approximation Guarantee*	*Complexity*	*Unweighted Only*	*No Pre-processing*	*Ref*
χ	$2-\frac{2}{\chi}$				1
Δ	$2-\frac{2}{\Delta}$	$O(\Delta n^2 \log n)$			2
	$2-\frac{3}{\Delta+2}$	$T(n,m)$	$\star$		3
	$2-\frac{\log\Delta+O(1)}{\Delta}$	$T(n,m)+S$	$\star$		4
δ	$2-\frac{2}{\delta_H+1}$	$O(\delta_H n^{\frac{3}{2}})$	$\star$		5
$D(G)$	$2-\frac{2}{D(G)+1}$	$T(n,m)$			6
Planar	$\frac{3}{2}$	$T(n,m)$			7
	$1+\epsilon$	$O(\frac{1}{\epsilon}8^{\frac{1}{\epsilon}}n)$	$\star$	$\star$	8
n	$2-\frac{1}{\sqrt{n}}$	$T(n,m)$			9
	$2-\frac{2\log n}{\log\log n}$	$T(n,m)$			10
	$2-\frac{2c}{n}$				11
	$2-\frac{2\log n}{\log\log n}$	$O(nm)$	$\star$		12
p-claw free	$2-\frac{1}{p-1}$	$T(n,m)$			13
	$(2-\frac{2}{p})$	$O(nm\log n+n^3)$	$\star$		14
	$2-\frac{\log p+O(1)}{p}$	$T(n,m)+S$	$\star$		15
$\underline{\delta}$	$\frac{2}{1+\underline{\delta}}$	$O(mn)$	$\star$	$\star$	16

Independent Set Problem					
Graph Parameter	*Approximation Guarantee*	*Complexity*	*Unweighted Only*	*No Pre-processing*	*Ref*
χ	$\frac{2}{\chi}$				17
Δ	$\frac{1}{\lceil\frac{\Delta+1}{3}\rceil}$	$O(\Delta)$		$\star$	18
	$\frac{3}{\Delta+2}$	O(m)	$\star$	$\star$	19
δ	$\frac{2}{\delta+1}$	$O(\delta n^{\frac{3}{2}})$	$\star$		20
	$\frac{5}{2\delta+3}$	$O(\delta n^{\frac{3}{2}})$	$\star$		21
	$\frac{2}{\delta+2}$	$O(m)$	$\star$	$\star$	22
$D(G)$	$\frac{2}{D(G)+1}$	$T(n,m)$			23
Planar	$\frac{1}{2}$	4-coloring			24
	$1-\epsilon$	$O(\frac{1}{\epsilon}8^{\frac{1}{\epsilon}}n)$	$\star$	$\star$	25
p-claw free	$\frac{1}{p-1}$	$O(n\log n+m)$			26
	$\max\{\frac{2}{p},\frac{k}{2}\alpha-\frac{n}{\alpha}(\frac{k}{2}-1)\}$	$O(n^3)$	$\star$	$\star$	27

Table 3.1: Approximation Results for Vertex Cover and Independent Set

3.8 INTEGER PROGRAMMING WITH TWO VARIABLES PER INEQUALITY

The analysis of integer programming with two variables per inequality (IP2 for short) provides insight as to why approximations to vertex cover, independent set and the 2SAT problems work. Moreover, any minimization IP2 is at least as hard to approximate as the vertex cover problem (Section 3.8.3). That means that such problems are Max-$\mathcal{SNP}$-hard and approximating them with a factor better than 2 will imply similar factor approximation for the vertex cover problem.

As this book goes to print we have discovered another problem that is a special case of IP2, the minimum satisfiability problem, where one seeks a minimum weight collection of clauses that are satisfied. We also found an extension of IP2, [Hoc96], that implies a 2-approximation for the feasible cut problem, and gives, in polynomial time, super optimal half integral solution for several other problems, including the sparsest cut (see Chapter 5 for a discussion of this problem).

3.8.1 THE HALF INTEGRALITY AND THE LINEAR PROGRAMMING RELAXATION

Many linear programming relaxations have solutions that are integer multiples of $\frac{1}{2}$. These include the vertex cover, independent set and the dual of matching. Here we show that the reason for this property lies with the formulation's structure of two variables per constraint.

As discussed later, many IP2 problems have LP relaxations whose solutions are *not* integer multiples of $\frac{1}{2}$. One way of deriving half integral solutions is to convert the system of constraint inequalities into a system of *monotone* inequalities where the conversion may map integer solution to half integers. A polynomial time algorithm (Hochbaum and Naor [HN94]) is then used for optimizing over a system of *monotone* inequalities in bounded integer variables. An inequality in two variables is called *monotone* if it is of the form

$$ax_{j_i} - bx_{k_i} \geq c$$

where a and b are both nonnegative. Although, as proved by Lagarias [Lag85], even the problem of finding a feasible solution of a system of monotone inequalities in integers is *NP*-complete, the algorithm of Hochbaum and Naor [HN94] finds an optimal solution in time $O(mnU^2 \log(Un^2/m))$ where U is the largest upper bound, i.e. in pseudo-polynomial time. For IP2s that include nonmonotone inequalities, we use a transformation of nonmonotone inequalities to monotone inequalities proposed by Edelsbrunner, Rote, and Welzl [ERW89]. The transformation does not preserve integrality, yet each solution to the transformed problem corresponds to a feasible solution of the original problem; and in addition it consists of integer multiples of $\frac{1}{2}$.

Consider a generic nonmonotone inequality of the form $ax + by \geq c$ where a and b are positive. (Any nonmonotone inequality can be written in this form, perhaps with a reversed inequality.) Replace each variable x by two variables, x^+ and x^-, and each inequality by two inequalities:

$$ax^+ - by^- \geq c$$
$$-ax^- + by^+ \geq c\ .$$

The two resulting inequalities are monotone. Note that upper and lower bounds constraints $\ell_j \leq x_j \leq u_j$ are transformed to

$$\ell_j \leq x_j^+ \leq u_j$$
$$-u_j \leq x_j^- \leq -\ell_j\ .$$

In the objective function, the variable x is substituted by $\frac{1}{2}(x^+ - x^-)$.

Monotone inequalities remain so by replacing the variables x and y in one inequality by x^+ and y^+, and in the second, by x^- and y^-, respectively. Note that the alternative formulation of the vertex cover problem that yields a minimum cut problem on a bipartite graph (presented in Section 3.7.1), is a special case of this transformation of nonmonotone to monotone inequalities.

Let $\mathcal{A}$ be the matrix of the constraints in the original system and let $\mathcal{A}^{(2)}$ be the matrix of the monotone system resulting from the above transformation. The matrix $\mathcal{A}^{(2)}$ consists of $2m$ inequalities with two variables per inequality, and $2n$ upper and lower bound constraints. The order of this matrix is therefore $(2m + 4n) \times 2n$.

We now sketch the algorithm of [HN94] which finds an optimal solution for an integer programming problem over monotone inequalities in time

$$O(mnU^2 \log(Un^2/\,m)).$$

Consider the optimization problem over a monotone system (IPM),

$$\text{(IPM)} \qquad \begin{array}{ll} \text{Min} & \sum_{j=1}^{n} w_j x_j \\ \text{subject to} & a_i x_{j_i} - b_i x_{k_i} \geq c_i \quad (i = 1, \ldots, m) \\ & \ell_j \leq x_j \leq u_j\ ,\ x_j \text{ integer} \quad (j = 1, \ldots, n), \end{array}$$

where a_i, b_i, c_i $(i = 1, \ldots, m)$, and w_j $(j = 1, \ldots, n)$ are rational, and ℓ_j and u_j $(j = 1, \ldots, n)$ are integers. The coefficients a_i and b_i $(i = 1, \ldots, m)$ are nonnegative but the objective function coefficients w_j $(j = 1, \ldots, n)$ may be negative. Note that we allow nonzero lower bounds on the variables that can be made nonnegative by translation.

A directed graph G is created where for each variable x_j in the interval $[\ell_j, u_j]$, there are $u_j - \ell_j + 1$ nodes representing it, one for each integer value in the range. A set of nodes is said to be *closed* if it contains all the nodes that can be reached via a directed path from any node in the set. It is shown that a maximum weight closed set in this graph corresponds to an optimal solution of (IPM). A section of the graph created is depicted in Figure 3.4

For each integer p in the range, there is an arc $(p, p-1)$ from the node representing the value p to the node representing the value $p-1$. The node representing ℓ_j has an arc directed to it from the source node s. Thus, if the source node is in a closed set then so are all ℓ_j nodes. The monotone inequalities are represented by arcs. For each potential value p of variable x_{p_i}, all inequalities in which x_{p_i} appears with a negative coefficient

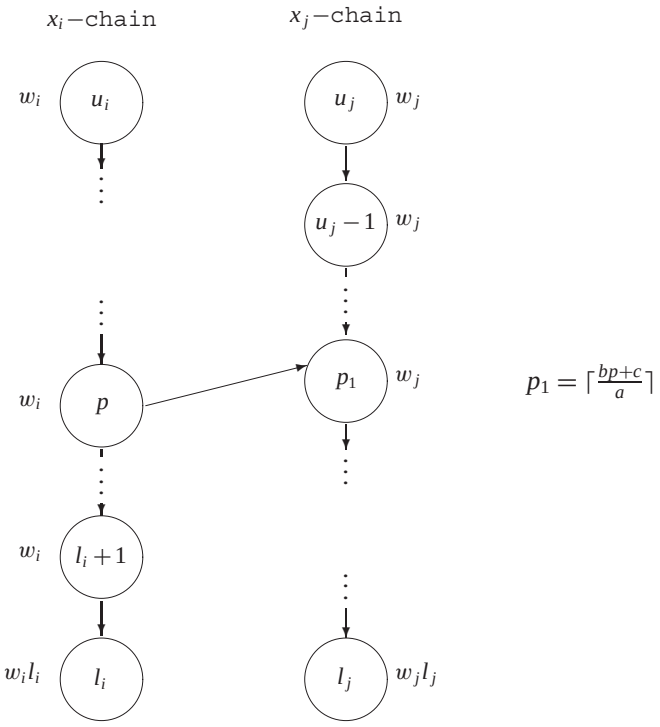

FIGURE 3.4

Representing the inequality $ax_i - bx_j \geq c$ between the chains for x_i and x_j.

impose a minimum value on the variable x_{j_i} that appears in the same inequality with a positive coefficient,

$$x_{j_i} \geq \left\lceil \frac{b_i p + c_i}{a_i} \right\rceil = p_1 \, .$$

This is represented by an arc going from node p of x_{p_i} to node p_1 of x_{j_i}. If $p_1 > u_{j_i}$, then the value p of the variable x_{p_i} is infeasible, and the upper bound of x_{p_i} is reset to $p-1$. A closed set containing s corresponds to a feasible solution to (IPM) where the variable x_j assumes the value of the largest node representing it in the closed set.

The nodes are now assigned weights as follows: node ℓ_j of variable x_j is assigned the value $-w_j \ell_j$, and all other nodes representing variable x_j are assigned the value $-w_j$. A maximum weight closed set corresponds then to an optimal solution to the minimization problem (IPM). The maximum closure in a graph is derived from solving a minimum cut problem in the graph after adding a source and a sink, placing arcs from the source to all nodes of positive weight with capacity equal to that weight, and placing arcs from all nodes with negative weight to the sink with capacity equal to the absolute value of that weight. All other arcs are assigned infinite capacity. The source

set of a minimum cut in this graph corresponds to a maximum weight closed set with the weights as specified. The justification for the algorithm of maximum closure is given by Picard [Pic7]. For the case of vertex cover, the algorithm reduces to the minimum cut in the bipartite network described in Figure 3.3 on page 114.

Whereas, for (VC) an optimal solution to the LP-relaxation of (VC) consists of integer multiples of $\frac{1}{2}$, such is not necessarily the case for IP2 where the use of the above transformation is necessary. IP2 problems may have LP relaxation solutions that are not integer multiple of $\frac{1}{2}$, and in fact it is even *NP*-hard to get an optimal solution among all those that are an integer multiple of $\frac{1}{2}$ as we demonstrate next.

Given a system of inequalities with two variables per inequality, let the set of feasible solutions for this system be

$$S = \{\mathbf{x} \in \mathbb{R}^n \mid \mathcal{A}\mathbf{x} \leq \mathbf{c}\} ,$$

and the feasible solutions to the monotone system resulting from the transformation above,

$$S^{(2)} = \{(\mathbf{x}^+, \mathbf{x}^-) \mid \mathcal{A}^{(2)}(\mathbf{x}^+, \mathbf{x}^-) \leq \mathbf{c}^{(2)} \ , \ \mathbf{x}^+, \mathbf{x}^- \in \mathbb{R}^n\} .$$

If $\mathbf{x} \in S$, $\mathbf{x}^+ = \mathbf{x}$, and $\mathbf{x}^- = -\mathbf{x}$, then $(\mathbf{x}^+, \mathbf{x}^-) \in S^{(2)}$. So, for every feasible solution in S, there exists a feasible solution in $S^{(2)}$. Conversely, if $(\mathbf{x}^+, \mathbf{x}^-) \in S^{(2)}$, then $\mathbf{x}^{(2)} = \frac{1}{2}(\mathbf{x}^+ - \mathbf{x}^-) \in S$. Hence, for every feasible solution in $S^{(2)}$, there is a feasible solution in S.

Let $S_I = \{\mathbf{x} \in S \mid \mathbf{x} \text{ integer }\}$, and let

$$S_I^{(2)} = \left\{ \frac{1}{2}(\mathbf{x}^+ - \mathbf{x}^-) \mid (\mathbf{x}^+, \mathbf{x}^-) \in S^{(2)} \text{ and } \mathbf{x}^+, \mathbf{x}^- \text{ integer} \right\} .$$

If $\mathbf{x} \in S_I$, then $\mathbf{x} \in S_I^{(2)}$. Thus, $S_I \subseteq S_I^{(2)} \subseteq S$.

In fact, the set of solutions $S_I^{(2)}$ is even smaller than the set of feasible solutions that are integer multiples of $\frac{1}{2}$. To see that, let

$$S^{(\frac{1}{2})} = \{\mathbf{x} \mid \mathcal{A}\mathbf{x} \leq \mathbf{c} \text{ and } \mathbf{x} \in \frac{1}{2} Z^n\} .$$

The claim is that $S_I^{(2)} \subset S^{(\frac{1}{2})}$, and $S^{(\frac{1}{2})}$ may contain points not in $S_I^{(2)}$. The following example illustrates such a case:

$$\begin{aligned} 5x + 2y &\leq 6 \\ 0 \leq x, y &\leq 1 . \end{aligned}$$

Obviously, $(x = 1,\ y = \frac{1}{2})$ is a feasible solution in $S^{(\frac{1}{2})}$. But there is no corresponding integer solution in $S_I^{(2)}$ as $x^+ = -x^- = 1$ implies that $y^+ = y^- = 0$. It follows that the bound derived from optimizing over $S_I^{(2)}$ is tighter than a bound derived from optimizing over $S^{(\frac{1}{2})}$. Not only is this latter optimization weaker, but it is also in general *NP*-hard as stated in the following Lemma (proved in [HMNT93]).

LEMMA 3.7 Minimizing over a system of inequalities with two variables per inequality for $\mathbf{x} \in \frac{1}{2} \cdot Z^n$, is *NP*-hard.

3.8.2 COMPUTING AN APPROXIMATE SOLUTION

Here we show how to obtain a 2-approximation for the optimum of a bounded integer program with two variables per inequality in $O(mnU^2 \log(Un^2/m))$ time. Assume that the given integer program has a feasible integer solution denoted by $z_1, \ldots, z_n$. (This can be tested in polynomial time as proved in Lemma 3.9.)

We first transform the integer program into a monotone integer system and compute an optimal solution for the monotone system as in the previous section. For every variable x_i $(i = 1, \ldots, n)$, let m_i^+ and m_i^- denote the respective values of x_i^+ and x_i^- in the optimal solution of the monotone system. For $i = 1, \ldots, n$, let $m_i^* = \frac{1}{2}(m_i^+ - m_i^-)$. We define the following solution vector, denoted by $\ell = (\ell_1, \ldots, \ell_n)$, where for $i = 1, \ldots, n$:

$$\ell_i = \begin{cases} \min\{m_i^+, -m_i^-\} & \text{if } z_i \le \min\{m_i^+, -m_i^-\}, \\ z_i & \text{if } \min\{m_i^+, -m_i^-\} \le z_i \le \max\{m_i^+, -m_i^-\}, \\ \max\{m_i^+, -m_i^-\} & \text{if } z_i \ge \max\{m_i^+, -m_i^-\}\,. \end{cases} \tag{3.9}$$

LEMMA 3.8 The vector ℓ is a feasible solution of the given integer program.

Proof. Let $ax_i + bx_j \ge c$ be an inequality where a and b are nonnegative. We check all possible cases. If ℓ_i is equal to z_i or $\min\{m_i^+, -m_i^-\}$, and ℓ_j is equal to z_j or $\min\{m_j^+, -m_j^-\}$, then clearly, $a\ell_i + b\ell_j \ge az_i + bz_j \ge c$. Suppose $\ell_i \ge z_i$ and $\ell_j = \max\{m_j^+, -m_j^-\}$. By construction, we know that

$$am_i^+ - bm_j^- \ge c \quad \text{and} \quad -am_i^- + bm_j^+ \ge c\,.$$

If $\ell_i \ge -m_i^-$, then, $a\ell_i + b\ell_j \ge -am_i^- + bm_j^+ \ge c$. Otherwise, $a\ell_i + b\ell_j \ge am_i^+ - bm_j^- \ge c$. The last case is when $\ell_i = \max\{m_i^+, -m_i^-\}$, and $\ell_j = \max\{m_j^+, -m_j^-\}$. In this case,

$$a\ell_i + b\ell_j \ge\ am_i^+ - bm_j^- \ge c\,.$$

The other types of inequalities are handled similarly. ■

We showed that vector ℓ is a feasible solution. We now argue that it also approximates the optimum.

THEOREM 3.7

1. The vector ℓ is a 2-approximate solution of the bounded integer program.
2. The value of the objective function at the vector $\mathbf{m}^*$ is at least a half of the value of the objective function of the best integer solution.

Proof. By construction, $\ell \le 2\mathbf{m}^*$. From the previous subsection we know that the vector $\mathbf{m}^*$ provides a lower bound on the value of the objective function for any integral solution. Hence, the theorem follows. ■

The complexity of the algorithm is dominated by the complexity of the procedure in [HN94] for optimizing over a monotone system. The running time is $O(mnU^2 \log(Un^2/m))$.

3.8.3 THE EQUIVALENCE OF IP2 TO 2-SAT AND 2-SAT TO VERTEX COVER

The integer programming problem in two variables per inequality is in fact *equivalent* to a 2-SAT problem. Hence, 2-SAT already captures all the interesting properties of IP2. This has also algorithmic consequences—implying a fast algorithm for finding a feasible solution to IP2. We also demonstrate a reduction of 2-SAT to (VC). That implies the "fixing variables" property for 2-SAT and therefore for IP2.

The equivalence to 2-SAT is shown using an idea of T. Feder: Recall that for each variable x_i we have $0 \le x_i \le u_i < \infty$ $(i = 1, \ldots, n)$. We replace each variable x_i by u_i binary variables $x_{i\ell}$ $(\ell = 1, \ldots, u_i)$, with the constraints $x_{i\ell} \ge x_{i,\ell+1}$ $(\ell = 1, \ldots, u_i - 1)$. Subject to these constraints, the correspondence between x_i and the u_i-tuple $(x_{i1}, \ldots, x_{iu_i})$ is one-to-one and is characterized by $x_{i\ell} = 1$ if and only if $x_i \ge \ell$ $(\ell = 1, \ldots, u_i)$, or, equivalently, $x_i = \sum_{\ell=1}^{u_i} x_{i\ell}$.

We now explain how to transform the constraints of the given system into constraints in terms of the $x_{i\ell}$'s. Suppose

$$a_{ki}x_i \; + \; a_{kj}x_j \; \ge \; b_k$$

is one of the given constraints. There are several cases to be distinguished. Without loss of generality, assume both a_{ki} and a_{kj} are nonzeros. Consider the case where both are positive, and assume without loss of generality that $0 < b_k < a_{ki}u_i + a_{kj}u_j$. For every ℓ $(\ell = 0, \ldots, u_i)$, let

$$\alpha_{k\ell} = \left\lceil \frac{b_k - \ell a_{ki}}{a_{kj}} \right\rceil - 1 \, .$$

It is easy to see that for an integer solution $\mathbf{x}$, $a_{ki}x_i + a_{kj}x_j \ge b_k$ if and only if for every ℓ $(\ell = 0, \ldots, u_i)$,

$$\text{either } \; x_i > \ell \; \text{ or } \; x_j > \alpha_{k\ell}$$

or, equivalently,

$$\text{either } \; x_i \ge \ell + 1 \; \text{ or } \; x_j \ge \alpha_{k\ell} + 1 \, .$$

Under the above transformation between the x_i's and the $x_{i\ell}$'s, this is equivalent to:

1. For every ℓ $(\ell = 0, 1, \ldots, u_i - 1)$, if $0 \le \alpha_{k\ell} < u_j$, then either $x_{i,\ell+1} = 1$ or $x_{j,\alpha_{k\ell}+1} = 1$, and if $\alpha_{k\ell} \ge u_j$, then $x_{i,\ell+1} = 1$.
2. For $\ell = u_i$, if $\alpha_{ku_i} \ge 0$, then $x_{j,\alpha_{ku_i}+1} = 1$ (since we have $\alpha_{ku_i} < u_j$).

The disjunction in (i) can be written as

$$x_{i,\ell+1} + x_{j,\alpha_{k\ell}+1} \ge 1 \, .$$

Thus, altogether we have replaced one original constraint on x_i and x_j by at most $u_i + 1$ constraints on the variables $x_{i\ell}$ and $x_{j\ell}$. The other cases, corresponding to different sign combinations of a_{ki}, a_{kj}, and b_k, can be handled in a similar way.

If the above transformation is applied to a monotone system of inequalities, then the resulting 2-SAT integer program is also monotone.

To summarize, we replace the n original variables and m original constraints by $\bar{u} = \sum_{j=1}^{n} u_j$ new variables and at most $mU + \bar{u}$ new constraints, where $U = \max_i u_i$.

The time bounds for finding a feasible solution are as follows.

LEMMA 3.9 A feasible solution to a bounded integer program with two variables per inequality can be computed in $O(m+n+\bar{u}+mU)$ time.

Proof. A feasible solution to a 2-SAT integer program can be found in linear time using the algorithm of [EIS76]. Encoding a bounded integer program as a 2-SAT integer program generates $\bar{u}$ variables and at most $mU+\bar{u}$ constraints. Hence, the time bound follows. ■

We now show that any 2-SAT is *equivalent* to a vertex cover problem, i.e. to a nonmonotone form of 2-SAT. This has been observed by Seffi Naor. Given a 2-CNF formula F. We compute the transitive closure of F, $T(F)$. This is done by repeating the following step until no more new clauses are generated: For every pair of clauses of the form $x \vee y$ and $\bar{x} \vee z$, add the clause $y \vee z$.

In an alternative approach for computing $T(F)$, one can create a directed graph. For each variable we have two nodes in the graph, one corresponding to the variable x, and the other to $\bar{x}$. Given a clause $x \vee y$, we replace it by two directed arcs, $\bar{y} \to x$ and $\bar{x} \to y$. We now consider the set of directed edges in the transitive closure of the graph. This transitive closure is symmetric and corresponds to the transitive closure of the set of clauses, F, by replacing every pair of symmetric directed arcs (of the form $\bar{y} \to x$ and $\bar{x} \to y$) by a clause.

Now generate a new undirected graph $G = (V, E)$ from the transitive closure of the clauses. For each variable we have two nodes in the graph, one corresponding to the variable x being true, and the other $\bar{x}$ corresponding to the variable x being false. Given a clause $x \vee y$ we place an edge $(x, y) \in E$. Finally, add the edges $(x, \bar{x})$ to the set E.

The claim is that a vertex cover in G corresponds to a satisfying assignment and vice versa. Obviously at least one of the endpoints of the edges of the form $(x, \bar{x})$ must be in the cover. We need to show that both x and $\bar{x}$ cannot be selected simultaneously in some optimal cover. Let $N(x)$ be the set of neighbors of x. Then the set of edges induced by $N(x)$ and $N(\bar{x})$ contains a complete bipartite graph, due to the transitive closure property. Any feasible vertex cover in a complete bipartite graph must contain at least one set of the bipartition. Therefore, either x or $\bar{x}$ are redundant in the cover and exactly one of the two is in the vertex cover. So, with the reduction above we showed that any 2-SAT problem on n variables and m clauses is equivalent to a vertex cover problem on $O(n)$ variables and $O(m^2)$ edges.

Since we established that IP2 is equivalent to 2-SAT, which in turn is equivalent to vertex cover (VC), it follows in particular that IP2 is equivalent to vertex cover. The 2-approximation to IP2 could hence be also deduced from this equivalence.

The solution to the relaxation of the vertex cover problem (VCR) has the "fixing variables" property that there exists an optimal solution that coincides with the relaxed solution in all integer components. The following lemma demonstrates that precisely the same idea applies to any integer programming problem IP2, after it is transformed to a 2-SAT.

With the reduction of 2-SAT to vertex cover, the direct proof to this lemma may be substituted by the corresponding lemma of Nemhauser and Trotter [NT75].

LEMMA 3.10 [HMNT93] Let $\mathbf{x}^{(2)}$ be an optimal solution of 2-SAT in the set $S_I^{(2)}$. Let

$$\text{INT} = \{j \mid x_j^{(2)} = 0 \text{ or } x_j^{(2)} = 1\} .$$

Then there is an optimal integer solution $\mathbf{z}$ of 2-SAT such that $z_j = x_j^{(2)}$ for $j \in \text{INT}$.

For the integer problem IP2 the fixing of variables imply that there exists an optimal solution $\mathbf{z}^*$ such that,

$$\min\{m_i^+, -m_i^-\} \leq z_i^* \leq \max\{m_i^+, -m_i^-\}.$$

If the upper and lower bounds are equal, then z_i may be fixed at that value. The interval gets smaller as more of the 2-SAT binary variables are fixed.

3.8.4 PROPERTIES OF BINARY INTEGER PROGRAMS

In this section we further investigate the properties of 2-SAT integer programs (or binary IP2s). We first consider the linear relaxation of a 2-SAT integer programming problem. It turns out that solutions of this relaxation always have denominators not greater than 2. Consequently, the basic solutions are integer multiples of $\frac{1}{2}$. This follows from the statement in the next lemma about the determinants of 2-SAT's *nonseparable* submatrices. A matrix is *nonseparable* if there is no partition of the columns and rows to two subsets (or more) C_1, C_2 and R_1, R_2 such that all nonzero entries in every row and column appear only in the submatrices defined by the sets $C_1 \times R_1$ and $C_2 \times R_2$. The following lemma applies only to nonseparable matrices, since one can construct a separable 2-SAT or (VC) matrix with an arbitrary number, K, of nonseparable ones on its diagonal, each of determinant 2. Thus we achieve a matrix with a determinant that is 2^K.

LEMMA 3.11 The determinants of all nonseparable submatrices of a 2-SAT linear programming problem have absolute value at most 2.

Proof. Let $\mathbf{A}$ denote the constraint matrix of a 2-SAT integer program. Thus, $\mathbf{A}$ has at most two non-zero entries in every row. We show that the absolute value of the determinant of any nonseparable square submatrix of $\mathbf{A}$ can be either 0, 1, or 2. The proof of this claim is by induction on the size of the submatrix. Since the entries of $\mathbf{A}$ are from $\{-1, 0, 1\}$, the claim holds for 1×1 submatrices. Assume it holds for any $(m-1) \times (m-1)$ submatrix and we show that the claim holds for any $m \times m$ submatrix.

We may assume that each row and column in $\mathbf{A}$ has exactly two non-zero entries. Otherwise, there must be a row or a column where all entries, possibly with the exception of one, are zero. In either case, we can apply the inductive assumption directly and prove the claim. Let $\mathbf{A}_{ij}$ denote the submatrix obtained by deleting the i'th row and the j'th column from $\mathbf{A}$.

Without loss of generality, we may assume that the two non-zero elements in row i of $\mathbf{A}$ are in columns i and $i+1$ (modulo m). (Due to the nonseparability of the submatrix, this can be achieved by appropriate row and column interchanges.) Hence,

$$\det(\mathbf{A}) = A[1,1] \cdot \det(\mathbf{A}_{11}) - (-1)^m A[m,1] \cdot \det(\mathbf{A}_{m1}) .$$

The absolute values of the determinants of $\mathbf{A}_{11}$ and $\mathbf{A}_{m1}$ are equal to 1, since both are triangular matrices with nonzero diagonal elements. Therefore, the absolute value of the determinant of $\mathbf{A}$ is at most 2. ■

An immediate corollary of Lemma 3.11 is the fact that the value of every variable in a basic solution of the 2-SAT linear program is in the set $\{0, \frac{1}{2}, 1\}$. Although for binary integer problems the subdeterminants can be of value greater than 2, and hence the solutions would not be in this set, we get rid of these "unnecessary" solutions by reducing the problem first to 2-SAT. In a 2-SAT system the variables are assumed to be binary. Lemma 3.11, however, applies to any linear programming problem with a constraint matrix with coefficients $0, 1, -1$, and at most two nonzero elements in each row. We call such a system *generalized* 2-SAT. Note that we do not assume the existence of finite upper bounds on the variables. We will show that a 2-approximation can be achieved even for such systems.

LEMMA 3.12 A generalized 2-SAT has the property that $S_I^{(2)} = S^{(\frac{1}{2})}$.

Proof. It suffices to prove that $S^{(\frac{1}{2})}$ is contained in $S_I^{(2)}$. Let $\mathbf{x} \in S^{(\frac{1}{2})}$. Define a solution $(\mathbf{x}^+, \mathbf{x}^-)$ as follows. For $j = 1, \ldots, n$,

1. If x_j is an integer, set $x_j^+ = -x_j^- = x_j$.

2. If x_j is a noninteger, then set $x_j^+ = x_j + \frac{1}{2}$ and $x_j^- = -x_j + \frac{1}{2}$.

It is easy to show that $(\mathbf{x}^+, \mathbf{x}^-)$ satisfies the (three) generic types of constraints defining $S_I^{(2)}$. For example, consider a constraint of the form $x_j^+ - x_k^- \geq c$. Since $\mathbf{x}$ is feasible, we have $x_j + x_k \geq c$. If either both x_j and x_k are integer or both are noninteger, then we have $x_j^+ - x_k^- = x_j + x_k \geq c$. Assuming that $x_j + x_k$ is noninteger, if $x_j + x_k \geq c$ then $x_j + x_k - \frac{1}{2} \geq c$. Using the fact that $x_j^+ \geq x_j$ and $-x_k^- \geq x_k - \frac{1}{2}$, it follows that $x_j^+ - x_k^- \geq x_j + x_k - \frac{1}{2} \geq c$. The other cases follow from similar considerations. ■

One corollary of Lemmas 3.11 and 3.12 is that the linear programming relaxation of a 2-SAT and a generalized 2-SAT can be solved by optimizing over the respective monotone system. Both problems are then solvable in strongly polynomial time: the 2-SAT as a maximum flow (or rather minimum cut) problem, and the generalized 2-SAT as a dual of a linear flow problem. Note that one could also solve these linear programs in strongly polynomial time without using the transformation to a monotone system by directly applying the algorithm of [Tar86]. The latter, however, is not as efficient as the best-known algorithms for solving maximum flow problems or linear flow problems.

We next show how to obtain a 2-approximation for a generalized 2-SAT integer program. First, we note that the procedure described above for IP2 is not applicable here since the variables might not have finite upper bounds. Since we already know how to solve the monotone system, the difficulty lies in finding a feasible integer solution or verifying that none exists. We perform this latter task as follows.

Let $(\mathbf{x}^+, \mathbf{x}^-)$ be an optimal solution of the monotone system, *i.e.*, $\mathbf{x} = \frac{1}{2}(\mathbf{x}^+ - \mathbf{x}^-)$ solves the linear programming relaxation. Using, if necessary, the transformation in the proof of Lemma 3.12, we may assume that $x_j^+ = -x_j^-$ or $x_j^+ = -x_j^- + 1$ $(j = 1, \ldots, n)$. Next, we apply Lemma 3.8 to conclude that the given generalized 2-SAT integer program

is feasible if and only if there exists a feasible rounding of **x**. The latter can be tested by the linear time algorithm in [EIS76]. Moreover, Theorem 3.8 ensures that if such a rounding exists, then it is a 2-approximation.

3.8.5 DUAL FEASIBLE SOLUTIONS FOR IP2

The approach described for devising a 2-approximation algorithm for IP2 is analogous to the preprocessing or dual optimal approach for the vertex cover. Exploring this analogy more carefully raises the question whether an analogue of the dual feasible approach could apply as well. The advantage would be to do away with the need to solve the minimum cut problem on a graph optimally (or solving the respective linear program).

This turns out to be possible by reducing IP2 to an equivalent vertex cover problem and then applying any dual-feasible algorithm. For instance, if we choose the algorithm of Bar-Yehuda and Even (dual-feasible I), its running time is linear in the number of edges in the resulting graph which is $O(m^2U^2)$. This represents a minor improvement (if at all) compared to the running time of the min-cut-based procedure, $O(mnU^2 \log(Un^2/m))$.

3.9 THE MAXIMUM COVERAGE PROBLEM AND THE GREEDY

Consider the *maximum coverage* problem. Given a set system $\mathcal{S}$ and a parameter k, the *maximum coverage* problem is to find k sets such that the total weight of elements covered is maximized. This problem is clearly *NP*-hard, as *set cover* is reducible to it.

For the maximum coverage problem we are interested in describing the performance of a simple greedy algorithm. Consider the performance of a greedy algorithm, as depicted in Figure 3.5, that selects k sets by iteratively picking the set that covers the maximum weight group of currently uncovered elements. The performance of the greedy heuristic when optimizing a submodular function has been studied by Nemhauser and Wolsey [NW72] and Conforti and Cornuejols [CC84]; Vohra and Hall [VH93] have

```
GREEDY ← ∅
for        l = 1 … k do
           select G_l ∈ S that maximizes wt(GREEDY ∪ G_l)
           GREEDY ← GREEDY ∪ G_l
end
output GREEDY
```

FIGURE 3.5

Greedy heuristic for the maximum coverage problem

explicitly noted that *maximum coverage* falls into this context. Applications of the *maximum coverage* problem can be found in Section 3.9.2; the method discussed next for dealing with this problem is from [HP94].

3.9.1 THE GREEDY APPROACH

We establish the following theorem regarding the quality of solution returned by the greedy heuristic, where wt(GREEDY) and wt(OPT) are the total weight of elements covered by the greedy solution and the optimal solution, respectively. G_l refers to the l'th set selected by the greedy algorithm.

THEOREM 3.8 $\text{wt(GREEDY)} \geq \left[1-\left(1-\frac{1}{k}\right)^k\right]\text{wt(OPT)} > (1-\frac{1}{e})\,\text{wt(OPT)}.$

Thus, the greedy heuristic is a $(1-\frac{1}{e})$-approximation algorithm for the *maximum coverage* problem.

In fact, there is an example given in [HP94], taken from [DJS93], that shows this bound to be tight.

In order to prove the bound, we need to establish two lemmas.

LEMMA 3.13 $\text{wt}(\cup_{i=1}^{l} G_i) - \text{wt}(\cup_{i=1}^{l-1} G_i) \geq \frac{\text{wt}(OPT)-\text{wt}(\cup_{i=1}^{l-1} G_i)}{k}$, for $l = 1, 2, \ldots, k$.

Proof. At least $\text{wt(OPT)} - \text{wt}(\cup_{i=1}^{l-1} G_i)$ worth of elements not covered by the first $(l-1)$ sets selected by the greedy heuristic are covered by the k sets of OPT. Hence, by the pigeonhole principle, one of the k sets in the optimal solution must cover at least $\frac{\text{wt(OPT)}-\text{wt}(\cup_{i=1}^{l-1} G_i)}{k}$ worth of these elements. Since G_l is a set that achieves maximum additional coverage, it must also. (Note that $\text{wt}(\cup_{i=1}^{l} G_i) - \text{wt}(\cup_{i=1}^{l-1} G_i)$ represents the additional coverage achieved by G_l.) ■

LEMMA 3.14 $\text{wt}(\cup_{i=1}^{l} G_i) \geq \left[(1-(1-\frac{1}{k})^l)\right]\text{wt(OPT)}$, for $l = 1, 2, \ldots, k$.

Proof. We proceed by induction on l. For $l = 1$, the result holds: $\text{wt}(G_1) \geq \frac{\text{wt}(OPT)}{k}$, from Lemma 3.13. Now,

$$\begin{aligned}
\text{wt}(\cup_{i=1}^{l+1} G_i) &= \text{wt}(\cup_{i=1}^{l} G_i) + (\text{wt}(\cup_{i=1}^{l+1} G_i) - \text{wt}(\cup_{i=1}^{l} G_i)) \\
&\geq \text{wt}(\cup_{i=1}^{l} G_i) + \frac{\text{wt(OPT)} - \text{wt}(\cup_{i=1}^{l} G_i)}{k} \\
&= (1-\frac{1}{k})\text{wt}(\cup_{i=1}^{l} G_i) + \frac{\text{wt(OPT)}}{k} \\
&\geq (1-\frac{1}{k})(1-(1-\frac{1}{k})^l)\,\text{wt(OPT)} + \frac{\text{wt(OPT)}}{k} \\
&= (1-(1-\frac{1}{k})^{l+1})\,\text{wt(OPT)},
\end{aligned}$$

where the first inequality comes from Lemma 3.13, and the second inequality is from the induction hypothesis. ■

Theorem 3.8 follows directly from Lemma 3.14 by letting $l = k$, and noting that because $\lim_{k\to\infty} 1 - \left(1 - \frac{1}{k}\right)^k = 1 - \frac{1}{e}$ and $1 - \left(1 - \frac{1}{k}\right)^k$ is decreasing, it follows that $1 - \left(1 - \frac{1}{k}\right)^k > 1 - \frac{1}{e} > .632$.

The quality of the greedy heuristic for the *maximum coverage* problem was also studied in [VH93] in the context of a "maximal covering location problem." By interpreting the problem as one of maximizing a submodular function, they were able to apply results from [CC84] and [NW72] to achieve the performance guarantee of Theorem 3.8 for their location problem. As discussed in [HP94], however, in some applications of the *maximum coverage* problem the sets in $\mathcal{S}$ may be implicitly defined rather than explicitly given. For instance, consider the problem of covering a maximum weight set of edges of a graph with k cutsets; this problem arises in an application involving the testing of printed circuit boards for short-circuits [Lou92]. The greedy heuristic entails selecting a maximum cut in a graph at each step, which is itself *NP*-hard!

Thus, in some cases finding the optimal set at a given stage may itself be *NP*-hard. Suppose, however, that a solution within a factor of β of the optimal is selected at each step of the greedy heuristic; what can we then say about the quality of solution returned by this greedy-like algorithm? By modifying the analysis used to prove Theorem 3.8, the following theorem from [HP94] can be obtained:

THEOREM 3.9 Suppose that a modified version of the algorithm of Figure 3.5 is run so that G_l, is a set that causes an increase in coverage that is within a factor β of the maximum increase possible rather than being the set that causes the maximum increase in overall coverage at iteration l. Then, the solution returned by the algorithm achieves coverage with weight within a factor of

$$(1 - \frac{\beta}{k})^k > 1 - \frac{1}{e^\beta}$$

of the optimal.

Proof. First of all, Lemma 3.13 can be easily generalized to establish that, for $l = 1, 2, \ldots, k$,

$$\text{wt}(\cup_{i=1}^{l} G_i) - \text{wt}(\cup_{i=1}^{l-1} G_i) \geq \beta \frac{\text{wt}(OPT) - \text{wt}(\cup_{i=1}^{l-1} G_i)}{k} .$$

Secondly, Lemma 3.14 can be easily generalized to establish that, for $l = 1, 2, \ldots, k$,

$$\text{wt}(\cup_{i=1}^{l} G_i) \geq \left[(1 - (1 - \frac{\beta}{k})^l) \right] \text{wt(OPT)} .$$

Then, the theorem follows by setting $l = k$ in the above result, and noting that $1 - (1 - \frac{\beta}{k})^k$, which is decreasing in k, has limit $1 - \frac{1}{e^\beta}$ as k approaches ∞. ■

For the maximum cut problem, there exists a polynomial algorithm with $\beta = .878\ldots$ (see [GW94]), yielding a polynomial time algorithm with an approximation ratio guarantee of $(1 - \frac{1}{e^\beta}) > .584$ for the circuit-testing application.[2]

[2]It has been pointed out by M. Goemans that for this problem of covering edges with k cut sets, employing an algorithm based on the method of conditional expectations provides a performance guarantee of $1 - \frac{1}{2^k}$. See Chapter 11 for more on the method of conditional expectation.

We next briefly describe some additional problems that are instances of the *maximum coverage* problem.

3.9.2 APPLICATIONS OF THE MAXIMUM COVERAGE PROBLEM

Problems, are considered from a variety of applications regarding covering graphs with subgraphs, packing, and fixed parameter combinatorial optimization.

- ***Covering Graphs by Subgraphs:*** As already mentioned, the problem of covering edges by cut-sets arises in testing printed circuit boards for short-circuits. At each stage of the greedy algorithm, we are to find a maximum cut in the graph consisting of those edges not already covered. Now, although MAX CUT is itself *NP*-hard (see [GJ79]), there is a simple greedy MAX CUT algorithm with $\beta = \frac{1}{2}$, and a more involved one with $\beta = \min_{0 \le \theta \le \pi} \frac{2}{\pi} \frac{\theta}{1-\cos\theta} = .878\ldots$ (see [GW94] and Chapter 11).

  Problems of covering the edges of a graph by subgraphs satisfying some particular structure arise in other contexts as well. For example, we may wish to cover the maximum weight set of edges in a graph G using k subgraphs from a class $\mathcal{R}$; $\mathcal{R}$ may consist of triangles or other small cliques (see [GHY94]), or spanning trees, etc. At each stage, an optimal structure from the class $\mathcal{R}$ is selected in the graph that is identical to G except that previously covered edges have weight 0.

  Consider a k-stage forestry problem in which a set of cells are to be harvested at each stage, under the restriction that no two adjacent cells can be harvested during a given stage; that is, at each stage an *Independent Set* set must be selected in a corresponding graph (see [BWE92]). At each stage, a maximum weight independent set is to be found. While the independent set problem is *NP*-hard for general graphs (indeed, guaranteeing a β-approximate solution for any fixed $\beta > 0$ is *NP*-hard), for certain classes of graphs (as we have seen) approximation algorithms of varying quality are available.

- ***Packing and Layout Problems:*** Now consider packing problems with a given set U of objects to pack. The common nature of these applications is that the objective is to pack the maximum weight set of objects into k identical *bins*. Our greedy approach is to pack, as best possible, a single bin at a time using objects of U not already packed. Some examples:

  1. *Circuit Layout and Design:* Recent advances in multilayer IC technology have led to design problems in which an optimal assignment of objects to layers is to be made. For example, it has been shown that the topological planar routing problem, in which a maximum weight set of nets is to be assigned to k given layers such that all nets assigned to a given layer can be routed without any two nets crossing each other, is *NP*-hard (see [CL90]). On the other hand, finding a maximum weighted subset of nets to assign to a *single* layer can be solved in polynomial time (see [Sup87]), yielding a $(1 - \left(1 - \frac{1}{k}\right)^k)$-approximation algorithm to the topological planar routing problem using the greedy heuristic.

2. *Scheduling:* Suppose we are given a set of jobs to assign to k identical machines, where each machine has a set of restrictions as to which jobs can be grouped together The goal is is to schedule the largest weight set of jobs to the k machines. How well a single machine can be packed in a stage of our greedy approach will depend on the restrictions as to what can be scheduled together on a machine, and depend, on the set of jobs to be scheduled.
3. *Logistics:* Consider a logistics problem in which k identical vehicles are to be packed with a maximum weight set of items for delivery to a common destination. Given a set of items to be delivered, each having a specified benefit, we attempt to pack the vehicles, one at a time, with items of maximum total benefit.

- ***Fixed Parameter Combinatorial Optimization:*** In some standard optimization problems, the goal is to cover all the elements in a set using the smallest number of subsets. We will look at the versions of these problems in which, given a fixed parameter k that limits the number of subsets that we can select, we wish to cover the maximum weight set of elements.

  For instance, consider *Vertex Cover* in which we need to cover the maximum weight set of edges in a graph G using k vertices. At each stage, we select a vertex that covers the maximum weight set of edges not previously covered.

  Other such $(1-\left(1-\frac{1}{k}\right)^k)$-approximation algorithms can be derived for fixed parameter versions of well-known combinatorial optimization problems such as *Dominating Set*, *Minimum Test Set*, *Hitting Set*, and *Minimum Test Collection*. Refer to the glossary and [GJ79] for complete specifications of these problems.

  We note that several location problems, in which the goal is to locate k facilities so that as many customers as possible can each be served within a prespecified cost, can be modeled as a fixed-parameter version of the *Dominating Set* problem. For example, a problem in which the goal is to locate k new facilities so as to maximize market share (see [MZH83]), can be modeled as such; while the results in [MZH83] give special cases of the problem that can be solved in polynomial time, our greedy approach provides an α_k-approximation algorithm for general instances of the problem considered. We also note that a similar problem concerning the optimal location of bank accounts was given in [CFN77]; indeed, they provide a greedy algorithm that is shown to have a α_k-approximation guarantee via a different, and more complicated, analysis than the one presented here.

REFERENCES

[AH77] K. Appel and W. Haken. Every planar map is four colorable. Part I: Discharging. *Illinois J. Math.* 21:429–490, 1977.

[AHK77] K. Appel, W. Haken, and J. Koch. Every planar map is four colorable. Part II: Reducibility," *Illinois J. Math.* 21:491–567, 1977.

[Baa78] S. Baase. *Computer Algorithms: Introduction to Design and Analysis*, Addison-Wesley, Reading, MA, 1978.

[BP76] E. Balas and M. Padberg. Set partitioning: A survey. *SIAM Review* 18: 710–760, 1976.

[BS69] M. Balinski and K. Spielberg. Methods for integer programming: algebraic, combinatorial and enumeration. *J. Aronofsky, editor, Progress in Operations Research, III* 295–292, 1969.

[Bak83] B. S. Baker. Approximation algorithms for *NP*-complete problems on planar graphs. In *Proceedings of the 24th Annual Symposium on Foundations of Computer Science*, IEEE, 265–273, 1983.

[BWE92] F. Barahona, A. Weintraub, and R. Epstein. Habitat dispersion in forest planning and the stable set problem. *Operations Research*, 40:14–21, 1992.

[BYE81] R. Bar-Yehuda and S. Even. A linear time approximation algorithm for the weighted vertex cover problem. *J. of Algorithms* 2:198–203, 1981.

[BE85] R. Bar-Yehuda and S. Even. A local-ratio theorem for approximating the weighted vertex cover problem. *Annals of Discrete Mathematics* 25:27–45, 1985.

[BGS95] M. Bellare, O. Goldreich, and M. Sudan. Free bits and nonapproximability. *Proceedings of the* 36*th Annual IEEE Symposium on Foundations of Computer Science (FOCS95)*. 422–431, 1995.

[Bro41] R. L. Brooks. On coloring the nodes of a network. *Proc. Cambridge Philos. Soc.* 37:194–197, 1941.

[CC84] M. Conforti and G. Cornuejols. Submodular functions, matroids and the greedy algorithm: tight worst-case bounds and some generalizations of the Rado-Edmonds theorem. *Discrete Applied Mathematics* 7:257–275, 1984.

[CFN77] G. Cornuejols, M. L. Fisher, and G. L. Nemhauser. Location of bank accounts to optimize float: an analytic study of exact and approximate alogrithms. *Management Science*, 23(8):789–810, 1977.

[CL90] J. Cong and C. L. Liu. On the k-layer Planar Subset and Via Minimization Problems. In *Proceedings of the European Design Automation Conference*, pages 459–463, 1990.

[CNS81] N. Chiba, T. Nishizeki and N. Saito. A linear 5-coloring algorithm of planar graphs. *J. of Algorithms* 2:317–327, 1981.

[CM91] E. Cohen and N. Megiddo. Improved algorithms for linear inequalities with two variables per inequality. In *Proceedings of the Twenty Third Symposium on Theory of Computing*, New Orleans, 145–155, 1991.

[CK75] N. Christofides and S. Korman. A computational survey of methods for the set covering problem. *Management Science* 21:591–599, 1975.

[Chv79] V. Chvátal. A Greedy Heuristic for the Set-Covering Problem *Math. of Oper. Res.* Vol. 4, 3, 233–235, 1979.

[Clar83] K. L. Clarkson. A modification of the Greedy algorithm for the vertex cover. *Info. Proc. Lett.* 16:23–25, 1983.

[DJS93] B. Dasgupta, R. Janardan, and N. Sherwani. On the greedy algorithm for a covering problem. Unpublished manuscript, February 1993.

[ERW89] H. Edelsbrunner, G. Rote, and E. Welzl. Testing the necklace condition for shortest tours and optimal factors in the plane. *Theoretical Computer Science* 66:157–180, 1989.

[Erd70] P. Erdös. On the Graph-Theorem of Turán. *Math. Lapok*, 21:249–251, 1970.

[EIS76] S. Even, A. Itai, and A. Shamir. On the complexity of timetable and multicommodity flow problems. *SIAM Journal on Computing* 5:691–703, 1976.

[F95] U. Feige. A threshold of $\ln n$ for approximating set cover. Manuscript, 1995.

[FNT74] D. R. Fulkerson, G. L. Nemhauser, and L. E. Trotter, Jr. Two computationally difficult set covering problems that arise in computing the 1-width incidence matrices of steiner triple systems. *Mathematical Programming Study* 2:72–81, 1974.

[GHY94] O. Goldschmidt, D. S. Hochbaum, and G. Yu. Approximation Algorithms for the k-clique covering problem. To appear *SIAM J. of Discrete Math*, 1994.

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability*, W. H. Freeman, San Francisco, 1979.

[GJS76] M. R. Garey, D. S. Johnson, and L. Stockmeyer. Some simplified *NP*-complete graph problems. *Theoret. Comput. Sci.* I 237–267, 1976.

[GN72] R. S. Garfinkel and G. L. Nemhauser. Optimal set covering: A survey. In *Perspectives on optimization: A collection of expository articles*, A. M. Geoffrion, ed., 164–183, 1972.

[GP92] D. Gusfield and L. Pitt. A bounded approximation for the minimum cost 2-SAT problem. *Algorithmica* 8:103–117, 1992.

[GT88] A. V. Goldberg and R. E. Tarjan. A new approach for the maximum flow problem. *J.of ACM* 35:921–940, 1983.

[GW94] M. X. Goemans and D. P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. Submitted to *Journal of the ACM*, 1994.

[H94] M. M. Halldórsson. Private communication, 1994.

[HR94] M. M. Halldórsson and J. Radhakrishnan. Greed is good: approximating independent sets in sparse and bounded-degree graphs. *Proceedings of 26th ACM Symposium on Theory of Computing*, 439–448, 1994.

[HH86] N. G. Hall and D. S. Hochbaum. A fast approximation algorithm for the multicovering problem. *Discrete Applied Mathematics* 15:35–40, 1986.

[Har69] F. Harary. *Graph Theory*, Addison-Wesley, Reading, MA, 1969.

[Hoc82] D. S. Hochbaum. Approximation algorithms for the set covering and vertex cover problems. *SIAM J. Comput.* 11(3) 1982, an extended version: W.P. #64-79-80, GSIA, Carnegie-Mellon University, April 1980.

[Hoc83] D. S. Hochbaum. Efficient bounds for the stable set, vertex cover and set packing problems. *Discrete Applied Mathematics* 6:243–254, 1983.

[Hoc96] D. S. Hochbaum. A framework for half integrality and 2-approximations with applications to feasible cut and minimum satisfiability. Manuspcript, 1996.

[HN94] D. S. Hochbaum and J. Naor. Simple and fast algorithms for linear and integer programs with two variables per inequality. *SIAM Journal on Computing*, 23(6) 1179–1192, 1994.

[HMNT93] D. S. Hochbaum, N. Megiddo, J. Naor and A. Tamir. Tight bounds and 2-approximation algorithms for integer programs with two variables per inequality. *Mathematical Programming* 62:69–83, 1993.

[HK73] J.E. Hopcroft and R.M. Karp. A $n^{\frac{5}{2}}$ algorithm for maximum matchings in bipartite graphs. *SIAM J. Comput.* 2:225–231, 1973.

[HP94] D. S. Hochbaum and A. Pathria. Analysis of the greedy approach in covering problems. Unpublished manuscript, 1994.

[HSVW90] J. M. Ho, M. Sarrafzadeh, G. Vijayan, and C. K. Wong. Layer Assignment for Multichip Modules. *IEEE Transactions on Computer-Aided Design*, 9:1272–12 77, 1990.

[Joh74] D. S. Johnson. Approximation Algorithms for Combinatorial Problems. *J. Comput. System Sci.*, 9:256–278, 1974.

[KZ95] M. Karpinski and A. Zelikovsky. Approximating dense cases of covering problems (preliminary draft). Manuscript, Sept. 1995.

[KVY94] S. Khuller, U. Vishkin, and N. Young. A primal-dual parallel approximation technique applied to weighted set and vertex cover. *J. of Algorithms*, 17(2):280–289, 1994.

[Lag85] J. C. Lagarias. The computational complexity of simultaneous diophantine approximation problems. *SIAM Journal on Computing* 14:196–209, 1985.

[Lor66] L. C. Lorentzen. Notes on covering of arcs by nodes in an undirected graph. *Technical Report ORC 66.16, University of California, Berkeley*, 1966.

[Lou92] R. Loulou. Minimal Cut Cover of a Graph with an Application to the Testing of Electronic Boards. *Operations Research Letters*, 12(5):301–306, 1992.

[Lov66] L. Lovász. On Decomposition of Graphs. *Studia Scientiarum Mathematicarum Hungarica* 1:237–238, 1966.

[Lov75a] L. Lovász. Three short proofs in graph theory. *J. Combin. Theory (B)* 19:269–271, 1975.

[Lov75] L. Lovász. On the Ratio of Optimal Integral and Fractional Covers. *Discrete Math.* 13 383–390, 1975.

[MMK79] R. E. Marsten, M. R. Muller, and C. L. Killion. Crew Planning at Flying Tiger: A successful application of integer programming. *Management Science* 25:1175–1183, 1979.

[MST80] D. Matula, Y. Shiloach, and R. Tarjan. Two linear-time algorithms for 5-coloring a planar graph. Stanford Department of Computer Science, Report No. STAN-CS-80-830, 1980.

[MS85] B. Monien and E. Speckenmeyer. Ramsey Numbers and an approximation algorithm for the vertex cover problem. *Acta Informatica* 22:115–123, 1985.

[MZH83] N. Megiddo, E. Zemel, and S. L. Hakimi. The maximum coverage location problem. *SIAM Journal of Algebraic and Discrete Methods*, 4(2):253–261, 1983.

[Meg83] N. Megiddo. Towards a genuinely polynomial algorithm for linear programming. *SIAM Journal on Computing* 12:347–353, 1983.

[NT75] G. L. Nemhauser and L. E. Trotter, Jr. Vertex packings: Structural properties and algorithms. *Mathematical Programming* 8:232–248, 1975.

[NW72] G. L. Nemhauser and L. Wolsey. Maximizing submodular set functions: formulations and analysis of algorithms. In *Studies of Graphs and Discrete Programming* North-Holland, Amsterdam, 279–301, 1972.

[Pad79] M. W. Padberg. Covering and packing and knapsack problems. *Annals of Discrete Mathematics* 4:265–287, 1979.

[Pic7] J. C. Picard. Maximal closure of a graph and applications to combinatorial problems. *Management Science* 22:1268–1272, 1976.

[Pul79] W. R. Pulleyblank. Minimum node covers and 2-bicritical graphs. *Mathematical Programming* 17:91–103, 1979.

[She83] J. B. Shearer. A note on the independence number of triangle-free graphs. *Discrete Mathematics* 46:(1983) 83–87.

[Sup87] K. Supowit. Finding a Maximum Planar Subset of a Set of Nets in a Channel. *IEEE Transactions on Computer-Aided Design*, 6:93–94, 1987.

[SW68] G. Szekeres and W. S. Wilf. An inequality for the chromatic number of a graph. *Combin. Theory* 4:1–3, 1968.

[Tar86] É. Tardos. A strongly polynomial algorithm to solve combinatorial linear programs. *Operations Research* 34:250–256, 1986.

[Tro85] L. E. Trotter. Discrete packing and covering. in, O'hEigeartaigh et al. 21–31, 1985.

[Tur41] P. Turán. An External Problem in Graph Theory. *Mat. Fiz. Lapok*, 48:436–452, 1941.

[VH93] R. V. Vohra and N. G. Hall. A probabilistic analysis of the maximal covering location problem. *Discrete Applied Mathematics* 43:175–183, 1993.

[VS81] R. van Slyke. Covering problems in CCCI systems. Report to the Air Force Office of Scientific Research, 1981.

[Wig83] A. Wigderson. Improving the performance guarantee for approximate graph coloring. *Journal of the ACM* 30:729–735, 1983.

[YG92] G. Yu and O. Goldschmidt. On locally optimal independent sets and vertex covers. *Technical Report ORP92-01: Graduate School in Operations Research The University of Texas at Austin*, 1992.