2 Basic Methodologies and Applications

Teofilo F. Gonzalez
University of California, Santa Barbara

2.1 Introduction

In Chapter 1 we presented an overview of approximation algorithms and metaheuristics. This serves as an overview of Parts I, II, and III of this handbook. In this chapter we discuss in more detail the basic methodologies and apply them to simple problems. These methodologies are restriction, greedy methods, LP rounding (deterministic and randomized), α vector, local ratio and primal dual. We also discuss in more detail inapproximability and show that the "classical" version of the traveling salesperson problem (TSP) is constant ratio inapproximable. In the last three sections we present an overview of the application chapters in Parts IV, V, and VI of the handbook.

2.2 Restriction

Chapter 3 discusses *restriction* which is one of the most basic techniques to design approximation algorithms. The idea is to generate a solution to a given problem P by providing an optimal or suboptimal solution to a subproblem of P. A subproblem of a problem P means restricting the solution space for P by disallowing a subset of the feasible solutions. The idea is to restrict the solution space so that it has some structure, which can be exploited by an efficient algorithm that solves the problem optimally or suboptimally. For this approach to be effective the subproblem must have the property that, for every problem instance, its optimal or suboptimal solution has an objective function value that is "close" to the optimal one for P. The most common approach is to solve just one subproblem, but there are algorithms where more than one subproblem is solved and then the best of the solutions computed is the solution generated. Chapter 3 discusses this methodology and shows how to apply it to several problems. Approximation algorithms based on this approach are discussed in Chapters 35, 36, 42, 45, 46, 54, and 73. Let us now discuss a scheduling application in detail. This is the scheduling problem studied by Graham [1,2].

2.2.1 Scheduling

A set of n tasks denoted by $T_1, T_2, \ldots, T_n$ with processing time requirements $t_1, t_2, \ldots, t_n$ have to be processed by a set of m identical machines. A partial order C is defined over the set of tasks to enforce a set of precedence constraints or task dependencies. The partial order specifies that a machine cannot commence the processing of a task until all of its predecessors have been completed. Each task T_i has to be processed for t_i units of time by one of the machines. A (nonpreemptive) schedule is an assignment of tasks to time intervals on the machines in such a way that (1) each task T_i is processed continuously for t_i units of time by one of the machines; (2) each machine processes at most one task at a time; and (3) the precedence constraints are satisfied. The *makespan* of a schedule is the latest time at which a task is being processed. The scheduling problem discussed in this section is to construct a minimum makespan schedule for a set of partially ordered tasks to be processed by a set of identical machines. Several limited versions of this scheduling problem has been shown to be NP-hard [3].

Example 2.1

The number of tasks, n, is 8 and the number of machines, m, is 3. The processing time requirements for the tasks, and the precedence constraints are given in Figure 2.1, where a directed graph is used to represent the task dependencies. Vertices represent tasks and the directed edges represent task dependencies. The integers next to the vertices represent the task processing requirements. Figure 2.2 depicts two schedules for this problem instance.

In the next subsection, we present a simple algorithm based on restriction to generate provable good solutions to this scheduling problem. The solution space is restricted to schedules without forced "idle time," i.e., each feasible schedule does not have idle time from the time at which all the predecessors of task T_i (in C) are completed to the time when the processing of task T_i begins, for each i.

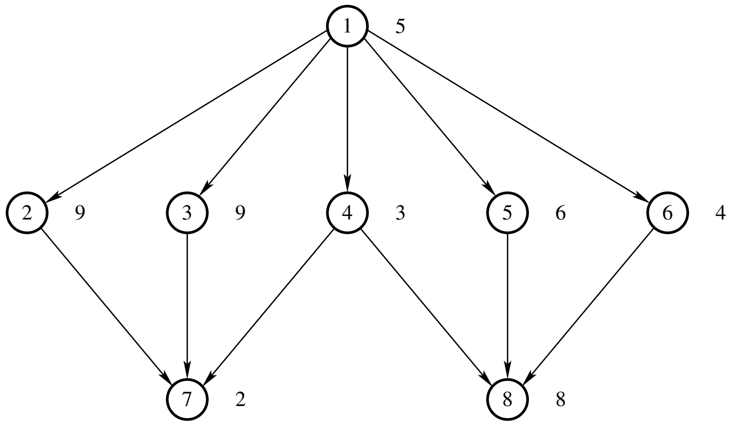

FIGURE 2.1 Precedence constraints and processing time requirements for Example 2.1.

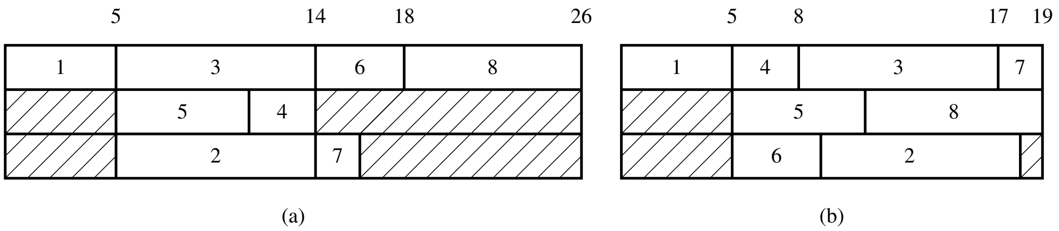

FIGURE 2.2 (a) and (b) represent two different AAT schedules for Example 2.1. Schedule (b) is a minimum makespan schedule.

2.2.2 Partially Ordered Tasks

Let us further restrict the scheduling policy to construct a schedule from time zero till all the tasks have been assigned. The scheduling policy is: whenever a machine becomes idle we assign one of the unassigned tasks that is ready to commence execution, i.e., we have completed all its predecessors. Any scheduling policy in this category can be referred to as a *no-additional-delay* scheduling policy. The simplest version of this scheduling policy is to assign any of the tasks (AAT) ready to be processed. A schedule generated by this policy is called an AAT schedule. These schedules are like the list schedules [1] discussed in Chapter 1. The difference is that list schedules have an ordered list of tasks, which is used to break ties. The analysis for both types of algorithms is the same since the list could be any list.

In Figure 2.2 we give two possible AAT schedules. The two schedules were obtained by breaking ties differently. The schedule in Figure 2.2(b) is a minimum makespan schedule. The reason for this is that the machines can only process one of the tasks T_1, T_5, or T_8 at a time, because of the precedence constraints.

Figure 2.2 suggests that an optimal schedule can be generated by just finding a clever method to break ties. Unfortunately, one cannot prove that this is always the case because there are problem instances for which all minimum makespan schedules are not AAT schedules.

The makespan of an AAT schedule is never greater than $2 - \frac{1}{m}$ times the one of an optimal schedule for the instance. This is expressed by

$$\frac{\hat{f}_I}{f_I^*} \leq 2 - \frac{1}{m}$$

where $\hat{f}_I$ is the makespan of any possible AAT schedule for problem instance I and f_I^* is the makespan of an optimal schedule for I. We establish this property in the following theorem:

Theorem 2.1

For every instance I of the identical machine scheduling problem, and every AAT schedule, $\frac{\hat{f}_I}{f_I^*} \leq 2 - \frac{1}{m}$.

Proof

Let S be any AAT schedule for problem instance I with makespan $\hat{f}_I$. By construction of the AAT schedules it cannot be that at some time $0 \leq t \leq \hat{f}_I$ all machines are idle. Let i_1 be the index of a task that finishes at time $\hat{f}_I$. For $j = 2, 3, \ldots,$ if task $T_{i_{j-1}}$ has at least one predecessor in C, then define i_j as the index of a task with latest finishing time that is a predecessor (in C) of task $T_{i_{j-1}}$. We call these tasks a *chain* and let k be the number of tasks in the chain. By the definition of task T_{i_j}, it cannot be that there is an idle machine from the time when task T_{i_j} completes its processing to the time when task $T_{i_{j-1}}$ begins processing. Therefore, a machine can only be idle when another machine is executing a task in the chain. From these two observations we know that

$$m\hat{f}_I \leq (m-1)\sum_{j=1}^{k} t_{i_j} + \sum_{j=1}^{n} t_j$$

Since no machine can process more than one task at a time, and since not two tasks, one of which precedes the other in C, can be processed concurrently, we know that an optimal makespan schedule satisfies

$$f_I^* \geq \frac{1}{m}\sum_{j=1}^{n} t_j \quad \text{and} \quad f_I^* \geq \sum_{j=1}^{k} t_{i_j}$$

Substituting in the above inequality, we know that $\frac{\hat{f}_I}{f_I^*} \leq 2 - \frac{1}{m}$. □

The natural question to ask is whether or not the approximation ratio $2 - \frac{1}{m}$ is the best possible for AAT schedules. The answer to this question is affirmative, and a problem instance for which this bound is tight is given in Example 2.2.

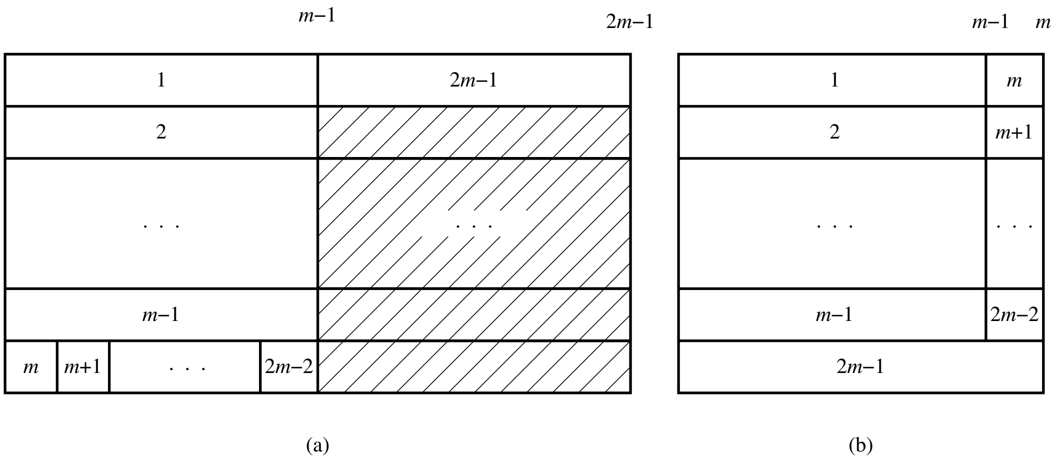

FIGURE 2.3 (a) AAT schedule. (b) Optimal schedule for Example 2.2.

Example 2.2

There are $2m - 1$ independent tasks. The first $m - 1$ tasks have processing time requirement $m - 1$, the next $m - 1$ tasks have processing time requirement one, and the last task has processing time requirement equal to m. An AAT schedule with makespan $2m - 1$ is given in Figure 2.3(a), and in Figure 2.3(b) we give a minimum makespan schedule.

Note that these results also hold for the list schedules [1] defined in Chapter 1. These type of schedules are generated by a no-additional-delay scheduling rule that is augmented by a list that is used to decide which of the ready-to-process tasks is the one to be assigned next.

Let us now consider the case when ties (among tasks that are ready) are broken in favor of the task with smallest index (T_i is selected before T_j if both tasks are ready to be processed and $i < j$). The problem instance I_A given in Figure 2.4 has three machines and eight tasks. Our scheduling procedure (augmented with a tie-breaking list) generates a schedule with makespan 14. In Chapter 1, we say that list schedules (which are this type of schedules) have anomalies. To verify this, apply the scheduling algorithm to instance I_A, but now there are four machines. One would expect a schedule for this new instance to have makespan at most 14, but you can easily verify that this is not the case. Now apply the scheduling algorithm to the instance I_A where every task has a processing requirement decreased by one unit. One would expect a schedule for this new instance to have makespan at most 14, but you can easily verify that is not the case. Apply the scheduling algorithm to the problem instance I_A without the precedence constraints from task

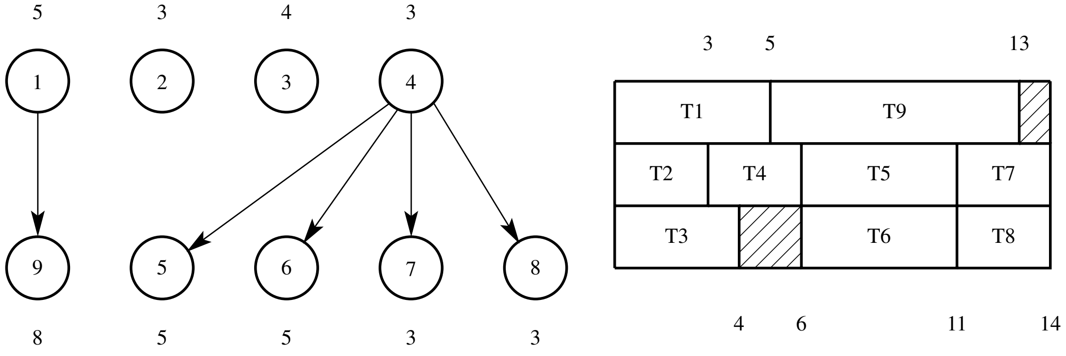

FIGURE 2.4 (a) Problem instance with anomalous behavior. (b) AAT schedule with tie-breaking list.

4 to 5 and task 4 to 6. One would expect a schedule for this instance to have makespan at most 14, but that is not the case. These are anomalies. Approximation algorithms suffer from this type of anomalous behavior. We need to be aware of this fact when using approximation algorithms.

As in the case of Example 2.2, the worst case behavior arises when the task with longest processing time is being processed while the rest of the machines are idle. Can a better approximation bound be established for the case when ties are broken in favor of a task with longest processing time (LPT)? The schedules generated by this rule are called *LPT schedules*. Any LPT schedule for the problem instance in Figure 2.3 is optimal. Unfortunately, this is not always the case and the approximation ratio in general is the same as the one for the AAT schedules. To see this just partition task $2m - 1$ in Example 2.2 (see Figure 2.3[a]) into a two-task chain. The first one has processing requirement of ϵ, for some $0 < \epsilon < 1$, and the second one $m - \epsilon$. The schedule generated by the LPT rule will schedule first all the tasks with processing requirement greater than 1 and then the two tasks in the chain.

The problem with the LPT rule is that it only considers the processing requirements of the tasks ready to process, but ignores the processing requirements of the tasks that follow it. We define the *weight of a directed path* as the sum of the processing time requirements of the tasks in the path. Any directed path that starts at task t with maximum weight among all paths that start at task t is called a *critical path for task t*. The critical-path (CP) schedule is defined as a no-additional-delay schedule where the decision of which task to process next is a task whose CP weight is largest among the ready-to-be processed tasks. The CP schedule is optimal for the problem instance that was generated by replacing the last task in Example 2.2 by two tasks. However, Graham constructed problem instances for which the makespan of the CP schedule is $2 - 1/m$ times the length of an optimal schedule.

It is not known whether or not a polynomial-time algorithm exists with a smaller approximation ratio even when the processing time requirements for all the tasks are identical and $m \geq 3$. There is a polynomial-time algorithm that generates an optimal schedule when $m = 2$, but the problem with different task processing times is NP-hard. In the next subsection we present an algorithm with a smaller approximation ratio for scheduling independent task.

2.3 Greedy Methods

Another way to generate suboptimal solutions is to apply greedy algorithms. The idea is to generate a solution by making a sequence of irrevocable decisions. Each of these decisions is a best possible choice at that point, for example, select an edge of least weight, select the vertex of highest degree, or select the task with longest processing time. Chapter 4 discusses greedy methods. The discussion also includes primal-dual approximation algorithms falling into this category. Chapter 5 discusses the recursive greedy method. This methodology is for the case when making the best possible decision is an NP-hard problem. A large portion of the bin packing algorithms are greedy algorithms. Bin packing and its variants are discussed in Chapters 32–35. Other greedy methods appear in Chapters 36, 38, 39, 44–46, 49, 50, 58, 59, and 69. Let us now discuss the LPT scheduling rule for scheduling independent tasks on identical machines.

2.3.1 Independent Tasks

Another version of this scheduling problem that has received considerable attention is when the tasks are independent, i.e., the partial order between the tasks is empty. Graham's [2] elegant analysis for LPT scheduling has become a classic. In fact, the analysis of quite a few subsequent exact and approximation scheduling algorithms follow the same approach.

First, we analyze the LPT scheduling rule. For this case there is only one possible schedule, modulo the relabeling of the tasks. We call this a "greedy method" because of the ordering of the tasks with respect to their processing requirements. This tends to generate schedules where the shortest tasks end up being processed last and the resulting schedules tend to have near-optimal makespan. However as we shall see, one may obtain the same approximation ratio by just scheduling the tasks using a list where the $2m$ task with longest processing time appear first (in sorted order) and the remaining tasks appear next in any order. This approach could be called "limited greedy." We discuss other approximation algorithms for this problem after presenting the analysis for LPT schedules.

Let I be any problem instance with n independent tasks and m identical machines. We use $\hat{f}_I$, as the makespan for the LPT schedule for I and f_I^* as the one for an optimal schedule. In the next theorem we establish the approximation ratio for LPT schedules.

Theorem 2.2

For every scheduling problem instance I with n independent tasks and m identical machines, every LPT schedule satisfies $\frac{\hat{f}_I}{f_I^*} \leq \frac{4}{3} - \frac{1}{3m}$.

Proof

It is clear that LPT schedules are optimal for $m = 1$. Assume that $m \geq 2$. The proof is by contradiction. Suppose the above bound does not hold. Let I be a problem instance with the least number of tasks for which $\frac{\hat{f}_I}{f_I^*} > \frac{4}{3} - \frac{1}{3m}$. Let n the number of tasks in I, m the number of machines, and assume that $t_1 \geq t_2 \geq \cdots \geq t_n$. Let k be the smallest index of a task that finishes at time $\hat{f}_I$. It cannot be that $k < n$, as otherwise the problem instance $T_1, T_2, \ldots, T_k$ is also a counterexample and it has fewer tasks than instance I, but by assumption problem instance I is a counterexample with the least number of tasks. Therefore, k must be equal to n.

By the definition of LPT schedules, we know that there cannot be idle time before task T_n begins execution. Therefore,

$$\sum_{i=1}^{n} t_i + (m-1)t_n \geq m\hat{f}_I$$

This is equivalent to

$$\hat{f}_I \leq \frac{1}{m}\sum_{i=1}^{n} t_i + \left(1 - \frac{1}{m}\right)t_n$$

Since each machine cannot process more than one task at a time, we know that $f_I^* \geq \sum_{i=1}^{n} t_i/m$. Combining these two bounds we have

$$\frac{\hat{f}_I}{f_I^*} \leq 1 + \left(1 - \frac{1}{m}\right)\frac{t_n}{f_I^*}$$

Since I is a counterexample for the theorem, this bound must be greater than $\frac{4}{3} - \frac{1}{3m}$. Simplifying we know that $f_I^* < 3t_n$. Since t_n is the task with smallest processing time requirement it must be that in an optimal schedule, for instance, I none of the machines can process three or more tasks. Therefore, the number of tasks n is at most $2m$.

For problem instance I, let S^* be an optimal schedule with least $\sum f_i^2$, where f_i is the makespan in S^* for machine i. Assume without loss of generality that the tasks assigned to each machine are arranged from largest to smallest with respect to their processing times. All machines have at most two tasks, as S^* is an optimal schedule for I which by definition is a counterexample for the theorem.

Let i and j be two machines in schedule S^* such that $f_i > f_j$, machine i has two tasks and machine j has at least one task. Let a and b be the task index for the last task processed by machine i and j, respectively. It cannot be that $t_a > t_b$, as otherwise applying the interchange given in Figure 2.5(a) results

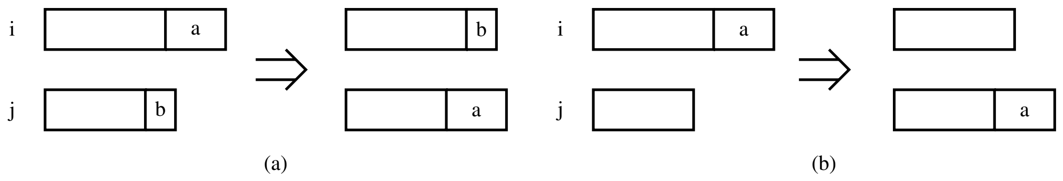

FIGURE 2.5 Schedule transformations.

in an optimal schedule with smaller $\sum f_i^2$. This contradicts the fact that S^* is an optimal schedule with least $\sum f_i^2$. Let i and j be two machines in schedule S^* such that machine i has two tasks. Let a be the task index for the last task processed by machine i. It cannot be that $f_i - t_a > f_j$ as otherwise applying the interchange given in Figure 2.5(b) results in an optimal schedule with smaller $\sum f_i^2$. This contradicts the fact that S^* is an optimal schedule with least $\sum f_i^2$.

Since the transformations given in Figure 2.5(a) and Figure 2.5(b) cannot apply, the schedule S^* must be of the form shown in Figure 2.6 after renaming the machines, i.e., machine i is assigned task T_i (if $i \leq n$) and task T_{2m-i+1} (if $2m - i + 1 \leq n$). But this schedule is an LPT schedule and $\hat{f} = f^*$. Therefore, there cannot be any counterexamples to the theorem. This completes the proof of the theorem. □

For all m there are problem instances for which the ratio given by Theorem 2.2 is tight. In Figure 2.7 we give one of such problem instance for three machines.

The important properties needed to prove Theorem 2.2 are that the longest $2m$ tasks need to be scheduled via LPT, and either the schedule will be optimal for the $2m$ task or at least three tasks will be assigned to a machine. The first set of m tasks, the ones with longest processing time, will be assigned to one machine each, so the order in which they are assigned is not really important. The next set of m tasks need to be assigned from longest to shortest processing times as in the LPT schedule. The remaining tasks can be assigned in any order as long as whenever a machine finishes a task the next task in the list is assigned to that machine. Any list schedule whose list follows the above ordering can be shown to have makespan at most $\frac{4}{3} - \frac{1}{3m}$ times the one of an optimal schedule. These type of schedules form a restriction on the solution space.

It is interesting to note that the problem of scheduling $2m$ independent tasks is an NP-hard problem. However, in polynomial time we can find out if there is an optimal schedule in which each machine has at most two tasks. And this is all that is needed to establish the $\frac{4}{3} - \frac{1}{3m}$ approximation ratio. One of the first

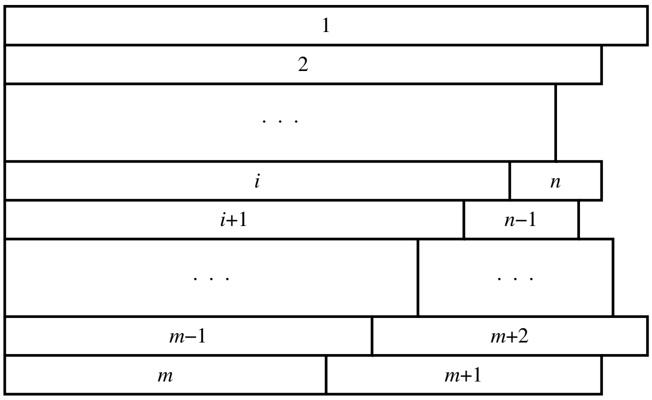

FIGURE 2.6 Optimal schedule.

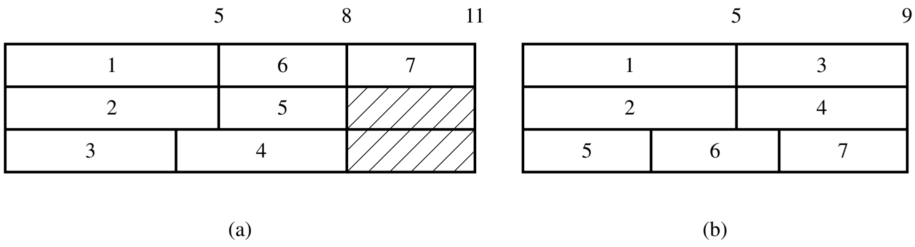

FIGURE 2.7 (a) LPT schedule. (b) Optimal schedule.

avenues of research explored was to see if the same approach would hold for the longest $3m$ tasks. That is, give a polynomial-time algorithm that finds an optimal schedule in which each machine has at most three tasks. If such an algorithm exists, we could use it to generate schedules that are within $\frac{5}{4} - \frac{1}{4m}$ times the makespan of an optimal schedule. This does not seem possible as Garey and Johnson [3] established that this problem is NP-hard.

Other approximation algorithms with improved performance were subsequently developed. Coffman et al. [4] introduced the multifit (MF) approach. A k attempt MF approach is denoted by MF_k. The MF_k procedure performs k binary search steps to find the smallest capacity c such that all the tasks can be packed into a set of m bins when packing using first fit with the tasks sorted in nondecreasing order of their processing times. The tasks assigned to bin i correspond to machine i and c is the makespan of the schedule. The approximation ratio has been shown to be $1.22 + 2^{-k}$ and the time complexity of the algorithm is $O(n \log n + kn \log m)$. Subsequent improvements to $1.2 + 2^{-k}$ [5] and $\frac{72}{61} + \frac{1}{2^k}$ [6] were possible within the same time complexity bound. However, the latter algorithm has a very large constant associated with the big "oh" bound.

Following a suggestion by D. Kleitman and D. E. Knuth, Graham [2] was led to consider the following scheduling strategy. For any $k \geq 0$ an optimal schedule for the longest k tasks is constructed and then the remaining tasks are scheduled in any order using the no-additional-delay policy. He shows that this algorithm has an approximation ratio $1 + \frac{1-1/m}{1+\lceil k/m \rceil}$ and takes $O(n \log m + km^k)$ time when there is a fixed number of machines. This was the first polynomial-time approximation scheme for any problem. This polynomial-time approximation scheme, as well as the ones for other problems are explained in more detail in Chapter 9. Fully polynomial-time approximation schemes are not possible for this problem unless $P = NP$ [3].

2.4 Relaxation: Linear Programming Approach

Let us now consider the minimum-weight vertex cover, which is a fundamental problem in the study of approximation algorithms. This problem is defined as follows

Problem: Minimum-weight vertex cover.
Instance: Given a vertex weighted undirected graph G with the set of vertices $V = \{v_1, v_2, \ldots, v_n\}$, edges $E = \{e_1, e_2, \ldots, e_m\}$ and a positive real number (weight) w_i assigned to each vertex v_i.
Objective: Find a minimum-weight vertex cover, i.e., a subset of vertices $C \subset V$ such that every edge is incident to at least one vertex in C. The weight of the vertex cover C is the sum of the weight of the vertices in C.

It is well known that the minimum-weight vertex cover problem is an NP-hard problem. Now consider the following simple greedy algorithm to generate a vertex cover. Assume without loss of generality that the graph G does not have isolated vertices, i.e., vertices without any edges. An edge is said to be *uncovered* with respect to a set of vertices C if both of its endpoints are vertices in $V \backslash C$, i.e., if both endpoints are not in C.

```
Algorithm Min-Weight(G)
    Let $C = \emptyset$;
    while there is an uncovered edge do
        Let $U$ be the set of vertices adjacent to at least one uncovered edge;
        Add to $C$ a least weight vertex in set $U$;
    endwhile
end
```

Algorithm `Min-Weight` is not a constant-ratio approximation algorithm for the vertex cover problem. Consider the family of star graphs $\mathcal{K}$ each with $l + 1$ nodes, l edges, the center vertex having weight k and the l leaves having weight 1, for any positive integers $k \geq 2$ and $l \geq 3$. For each of these graphs Algorithm `Min-Weight` generates a vertex cover that includes all the leaves in the graph and the weight of the cover is l. For all graphs in $\mathcal{K}$ with $k = 2$, an optimal cover has weight 2 and includes only the center vertex. Therefore, Algorithm `Min-Weight` has an approximation ratio of at least $l/2$, which cannot be bounded above by any fixed constant.

Algorithm `Max-Weight` is identical to Algorithm `Min-Weight`, but instead of selecting the vertex in set U with least weight, it selects one with largest weight. Clearly, this algorithm constructs an optimal cover for the graphs identified above where Algorithm `Min-Weight` performs badly. For every graph in $\mathcal{K}$, this algorithm selects as its vertex cover the center vertex which has weight k. Now for all graphs in $\mathcal{K}$ with $l = 2$, an optimal cover consists of both leaf vertices and it has weight 2. Therefore, the approximation ratio for Algorithm `Max-Weight` is at least $k/2$, which cannot be bounded above by any fixed constant.

All of the graphs identified above, where one of the algorithms performs badly, have the property that the other algorithm constructs an optimal solution. A compound algorithm that runs both algorithms and then selects the better of the two vertex covers may be a constant-ratio algorithm for the vertex cover problem. However, this compound algorithm can also be easily fooled by just using a graph consisting of two stars, where each of the individual algorithms failed to produce good solutions. Therefore, this compound algorithm fails to generate constant-ratio approximate solutions. One may now argue that we could partition the graph into connected components and apply both algorithms to each component. For these "two-star" graphs the new compound algorithm will generate an optimal solution. But in general this new approach fails to produce a constant-ratio approximate solution for all possible graphs. Adding an edge between the center vertex in the "two-star" graphs gives rise to problem instances for which the new compound algorithm fails to provide a constant ratio approximate solution.

A more clever approach is a modified version of Algorithm `Min-Weight`, where instead of selecting a vertex of least possible weight in set U, one selects a vertex v in set U with least $w(v)/u(v)$, where $u(v)$ is the number of uncovered edges incident to vertex v. This seems to be a better strategy because when vertex v is added to C it covers $u(v)$ edges at a total cost of $w(v)$. So the cost (weight) per edge of $w(v)/u(v)$ is incurred when covering the uncovered edges incident to vertex v. This strategy solves optimally the star graphs in $\mathcal{K}$ defined above. However, even when all the weights are equal, one can show that this is not a constant ratio approximation algorithm for the weighted vertex cover problem. In fact, the approximation ratio for this algorithm is about $\log n$. Instances with a simple recursive structure that asymptotically achieve this bound as the number of vertices increases can be easily constructed. Chapter 3 gives an example of how to construct problem instances where an approximation algorithm fails to produce a good solution.

Other approaches to solve the problem can also be shown to fail to provide a constant-ratio approximation algorithm for the weighted vertex cover. What type of algorithm can be used to guarantee a constant-ratio solution to this problem? Let us try another approach.

Another way to view the minimum-weight vertex cover is by defining a 0/1 variable x_i for each vertex v_i in the graph. The 0/1 vector X defines a subset of vertices C as follows. Vertex v_i is in C if and only if $x_i = 1$. The set of vertices C defined by X is a vertex cover if and only if for every edge $\{i, j\}$ in the graph $x_i + x_j \geq 1$. The vertex cover problem is expressed as an instance of the 0/1 integer linear programming (ILP) as follows:

$$\begin{aligned} \text{minimize} \quad & \sum\nolimits_{i \in V} w_i x_i & (2.1)\\ \text{subject to} \quad & x_i + x_j \geq 1 \quad \forall \{i, j\} \in E & (2.2)\\ & x_i \in \{0, 1\} \quad \forall i \in V & (2.3) \end{aligned}$$

The 0/1 ILP is also an NP-hard problem.

An important methodology for designing approximation algorithms is relaxation. In this case one relaxes the integer constraints for the x_i values. That is, we replace constraint (2.3) ($x_i = \{0, 1\}$) by $0 \leq x_i \leq 1$ (or simply $x_i \geq 0$, which in this case is equivalent). This means that we are augmenting the solution space by adding solutions that are not feasible for the original problem. This approach will at least provide us with what appears to be a good lower bound for the value of an optimal solution of the original problem, since every feasible solution to the original problem is a feasible solution to the relaxed problem (but the converse is not true). This relaxed problem is an instance of the linear programming (LP) problem which can be solved in polynomial time. Let X^* be an optimal solution to the LP problem. Clearly, X^* might not be a vertex cover as the x_i^* values may be noninteger. The previous interpretation for the X^* values has been lost because it does not make sense to talk about a fractional part of a vertex being part of a vertex cover. To circumvent this situation, we need to use the X^* vector to construct a 0/1 vector $\hat{X}$ that represents a vertex cover. For a vector $\hat{X}$ to represent a vertex cover it needs to satisfy inequality (2.2) (i.e., $\hat{x}_i + \hat{x}_j \geq 1$), for every edge $e_k = \{i, j\} \in E$. Clearly, the inequalities hold for X^*. This means that for each edge $e_k = \{i, j\} \in E$ at least one of x_i^* or x_j^* has value at least greater than or equal to $\frac{1}{2}$. So the vector $\hat{X}$ defined from X^* as $\hat{x}_i = 1$ if $x_i^* \geq \frac{1}{2}$ (rounding up) and $\hat{x}_i = 0$ if $x_i^* < \frac{1}{2}$ (rounding down) represents a vertex cover. Furthermore, because of the rounding up the objective function value for the vertex cover $\hat{X}$ is at most $2\sum w_i x_i^*$. Since $\sum w_i x_i^*$ value is a lower bound for an optimal solution to the weighted vertex cover problem, we know that this procedure generates a vertex cover whose weight is at most twice the weight of an optimal cover, i.e., it is a 2-approximation algorithm. This process is called (deterministic) *LP rounding*. Chapters 6, 7, 9, 11, 37, 45, 57, 58, and 70 discuss and apply this methodology to other problems.

Another way to round is via randomization, which means in this case that we flip a biased coin (with respect to x_i^* and perhaps other factors) to decide the value for $\hat{x}_i$. The probability of $\hat{X}$ is a vertex cover and its expected weight can be computed. By repeating this randomization process several times, one can show that a cover with weight at most twice the optimal one will be generated with very high probability. In this case it is clear that randomization is not needed. However, for other problems it is justified. Chapters 4, 6, 7, 11, 12, 57, 70, and 80 discuss LP randomized rounding, and Chapter 8 discusses more complex randomized rounding for semidefinite programming (SDP).

The above rounding methods have the disadvantage that an LP problem needs to be solved. Experimental evaluations over several decades have shown that the Simplex method solves quickly (in poly time) the LP problem. But the worst-case time complexity is exponential with respect to the problem size. In Chapter 1 we have discussed the Ellipsoid algorithm and more recent ones that solve LP problems. Even though these algorithms have polynomial-time complexity, there is a term that depends on the number of bits needed to represent the input. Much progress has been made in speeding up these procedures, but the algorithms are not competitive with typical $O(n^2)$ time algorithms for other problems.

Let us now discuss another approximation algorithm for the minimum vertex cover problem that it is "independent" of LP, and then we discuss a local-ratio and a primal-dual approach to this problem.

We call this approach the *α-vector* approach. For every vertex $i \in V$, we define $\delta(i)$ as the set of edges incident to vertex i. Let $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_m)$ be any vector of m nonnegative real values, where $m = |E|$ is the number of edges in the graph. For all k multiply the kth edge inequality by α_k,

$$\alpha_k x_i + \alpha_k x_j \geq \alpha_k \quad \forall e_k = \{i, j\} \in E \tag{2.4}$$

The total sum of these inequalities can be expressed as

$$\sum_{i \in V} \sum_{e_k \in \delta(i)} \alpha_k x_i \geq \sum_{e_k \in E} \alpha_k \tag{2.5}$$

Define $\beta_i = \sum_{e_k \in \delta(i)} \alpha_k$ for every vertex $i \in V$. In other words, β_i be the sum of the α values of all the edges incident to vertex i. Substituting in the above inequality we know that

$$\sum_{i \in V} \beta_i x_i \geq \sum_{e_k \in E} \alpha_k \tag{2.6}$$

Suppose that the α vector is such that $w_i \geq \beta_i$ for all i. Then it follows that

$$\sum_{i \in V} w_i x_i \geq \sum_{i \in V} \beta_i x_i \geq \sum_{e_k \in E} \alpha_k \tag{2.7}$$

In other words any vector α such that the resulting vector β computed from it satisfies $w_i \geq \beta_i$ provides us with the lower bound $\sum_{e_k \in E} \alpha_k$ for the objective function value of every vector X that represents a vertex cover. In other words, if we assign a positive weight to each edge in such a way that the sum of the weight of the edges incident to each vertex i is at most w_i, then the sum of the weight of the edges is a lower bound for an optimum solution.

This is a powerful lower bound. To get maximum strength we need to find a vector α such that $\sum_{e_k \in E} \alpha_k$ is maximum. But finding this vector is as hard as solving the LP problem described earlier. What if we find a maximal vector α, i.e., a vector that cannot possibly be increased in any of its components? This is a simple task. It is just a matter of starting with an α vector with all entries being zero and then increasing one of its components until it is no longer possible to do so. We keep on doing this until there are no edges whose α value can be increased. In this maximal solution, we know that for each edge in the graph at least one of its endpoints has the property that $\beta_i = w_i$, as otherwise the maximality of α is contradicted. Define the vector $\hat{X}$ from the α vector as follows: $x_i = 1$ if $\beta_i = w_i$, and $x_i = 0$, otherwise. Clearly, $\hat{X}$ represents a vertex cover because for every edge in the graph we know that at least one of its vertices has $\beta_i = w_i$. What is the weight of the vertex cover represented by $\hat{X}$? We know that $\sum w_i \hat{x}_i = \sum \beta_i \hat{x}_i \le 2 \sum \alpha_k$ because each α_k can contribute its value to at most two β_is. Therefore, we have a simple 2-approximation algorithm for the weighted vertex cover problem. Furthermore, the procedure to construct the vertex cover takes linear time with respect to the number of vertices and edges in the graph.

This algorithm was initially developed by Bar-Yehuda and Even [7] using the LP relaxation and its dual. This approach is called the *primal-dual* approach. It will be discussed later in this section. The above algorithm can be proven to be a 2-approximation algorithm without using the ILP formulation. That is, the same result can be established by just using simple combinatorial arguments [8].

Another related approach, called *local ratio*, was developed by Bar-Yehuda and Even [9]. Initially, each vertex is assigned a cost which is simply its weight and it is referred to as the *remaining cost*. At each step the algorithm makes a "down payment" on a pair of vertices. This has the effect of decreasing the remaining cost of each of the two vertices. Label the edges in the graph $\{e_1, e_2, \ldots, e_m\}$. The algorithm considers one edge at a time using this ordering. When the kth edge $e_k = \{i, j\}$ is considered, define γ_k as the minimum of the remaining cost of vertex i and vertex j. The edge makes a down payment of γ_k to each of its two endpoints and each of the two vertices has its remaining cost decreased by γ_k. The procedure stops when we have considered all the edges. All the vertices whose current cost is zero have been paid for completely and they are yours to keep. The remaining ones have not been paid for and there are "no refunds" (not even if you talk to the store manager). The vertices that have been paid for completely form a vertex cover. The weight of all the vertices in the cover generated by the procedure is at most twice $\sum_{e_k \in E} \gamma_k$, which is simply the sum of the down payments made. What is the weight of an optimal vertex cover? The claim is it is equal to $\sum_{e_k \in E} \gamma_k$. The reason is simple. Consider the first step when we introduce γ_1 for edge e_1. Let I_0 be the initial problem instance and I_1 be the resulting instance after deleting edge e_1 and reducing the cost of the two endpoints of edge e_1 by γ_1. One can prove that $f^*(I_0) = f^*(I_1) + \gamma_1$, and inductively that $f^*(I_0) = \sum_{e_k \in E} \gamma_k$ [10]. The algorithm is a 2-approximation algorithm for the weighted vertex cover. The approach is called *local ratio* because at each step one adds $2\gamma_k$ to the value of the solution generated and one accounts for γ_k value of an optimal solution. This local-ratio approach has been successfully applied to quite a few problems. The best feature of this approach is that it is very simple to understand and does not require any LP background.

The primal-dual approach is similar to the previous ones, but it uses the foundations of LP theory. The LP relaxation problem is

$$\text{minimize} \quad \sum_{i \in V} w_i x_i \tag{2.8}$$

$$\text{subject to} \quad x_i + x_j \ge 1 \quad \forall e_k = \{i, j\} \in E \tag{2.9}$$

$$x_i \ge 0 \qquad \forall i \in V. \tag{2.10}$$

The LP problem is called the *primal* problem. The corresponding dual problem is

$$\text{maximize} \quad \sum_{e_k \in E} y_k \tag{2.11}$$

$$\text{subject to} \quad \sum_{e_k \in \delta(i)} y_k \le w_i \quad \forall i \in V \tag{2.12}$$

$$y_k \ge 0 \qquad \forall e_k \in E \tag{2.13}$$

As you can see the Y vector is simply the α vector defined earlier, and the dual is to find a Y vector with maximum $\sum_{i \in V} y_i$. Linear programming theory [11,12] states that any feasible solution X to the primal problem and any feasible solution Y to the dual problem are such that

$$\sum_{e_k \in E} y_k \leq \sum_{i \in V} w_i x_i$$

This is called *weak duality. Strong duality* states that

$$\sum_{e_k \in E} y_i^* = \sum_{i \in V} w_i x_i^*$$

where X^* is an optimal solution to the primal problem and Y^* is an optimal solution to the dual problem. Note that the dual variables are multiplied by weights which are the right-hand side of the constraints in the primal problem. In this case all of them are one.

The primal-dual approach is based on the weak duality property. The idea is to first construct a feasible solution to the dual problem. That solution will give us a lower bound for the value of an optimal vertex cover, in this case. Then we use this solution to construct a solution to the primal problem. The idea is that the difference of the objective function value between the primal and dual solutions we constructed is "small." In this case we construct a maximal vector Y (as we did with the α vector before). Then we note that since the Y vector is maximal, then for at least one of the endpoints (say i) of every edge must satisfy Inequality 2.12 tight, i.e., $\sum_{e_k \in \delta(i)} y_k = w_i$. Now define vector X with $x_i = 1$ if inequality (2.12) is tight in the dual solution. Clearly, X represents a feasible solution to the primal problem and its objective function value is at most $2\sum_k y_k$. It then follows by weak duality that an optimal weighted vertex cover has value at least $\sum_k y_k$ and we have a 2-approximation algorithm for the weighted vertex cover. It is simple to see that the algorithm takes linear time (with respect to the number of vertices and edges in the graph) to solve the problem.

There are other ways to construct a solution to the dual problem. In Chapters 4 and 13 another method is discussed for finding a solution to the dual problem. Note the difference in the time required to construct the solution. Chapter 13 discusses a "distributed" version of this algorithm. This algorithm makes decisions by using only "local" information. Chapters 37, 39, 40, and 71 discuss several approximation algorithms based on variations of the primal dual approach. Some of these methods are not exactly primal dual, but may be viewed this way.

Linear programming has also been used as a tool to compute the approximation ratio of some algorithms. This type of research may eventually be called the *automatic analysis of approximation algorithms.* Chapter 3 discusses an early approach to compute the approximation ratio, and Chapter 39 discusses a more recent one. In the former case, a set of LP needed to be solved. Once this was computed it gave the necessary insight on how to prove it analytically. In the latter case, one just formulates the problem and finds bounds for the value of an optimal solution to the LP problem.

2.5 Inapproximability

Sahni and Gonzalez [13] established that constant-ratio polynomial time approximation algorithms exist for some problems only if $P = NP$. In other words, finding a suboptimal solution to some problems is as hard as finding an optimal solution. Any polynomial-time algorithm that generates k-approximate solution can be used to find an optimal solution to the problem in polynomial-time. One of these problems is the "classical" version of the TSP defined in Chapter 1, not the restricted one defined over metric graphs. To prove this result we show that an NP-complete problem, called the Hamiltonian Cycle (HC) problem, can be solved in polynomial time if there is a polynomial-time algorithm for the TSP that generates a k-approximate solution, for any fixed constant k. The HC problem is given an undirected graph, $G = (V, E)$, determine whether on not the graph has a HC. A HC for an undirected graph G is a path that starts at vertex 1, visits each vertex *exactly* once, and ends at vertex 1.

To prove this result a polynomial transformation (Chapter 1 and [3]) is used. Let $G = (V, E)$ be any instance of the HC problem with $n = |V|$. Now construct an instance $G' = (V', E', W')$ of the TSP as follows. The graph G' has n vertices and it is complete (all the edges are present). The edge $\{i, j\}$ in E' has weight 1 if the edge $\{i, j\}$ is in E, and weight Z otherwise. The value of Z is $(k-1)n + 2 > 1$. It will be clear later on why it was defined this way. If the graph G has a HC, then we know that the graph G' has a tour with weight n. However, if G does not have a HC, then all tours for the graph G' have weight greater than or equal to $n - 1 + Z$. A k-approximate solution (tour) when $f^*(G') = n$ must have weight at most $\hat{f}(G') \leq kf^*(G') = kn$. When G does not have a HC, the best possible tour that can be found by the approximation algorithm is one with weight at least $n - 1 + Z = kn + 1$. Therefore, if the approximation algorithm returns a tour with weight at most kn, then G has a HC, otherwise (the tour returned has weight $> kn$) G does not have a HC. Since the algorithm takes polynomial-time with respect to the number of vertices and edges in the graph, it then follows that the algorithm solves in polynomial time the HC problem. So we say that the TSP is inapproximable with respect to any constant ratio. It is inapproximable in the sense that a polynomial-time constant-ratio approximation algorithm implies the solution to a computational complexity question. In this case it is the $P = NP$ question.

In the last 15 years there have been new inapproximability results. These results have been for constant, $\ln n$, and n^ϵ approximation ratios. The techniques to establish some of these results are quite complex, but an important component continues to be reducibility. Chapter 17 discusses all of this work in detail.

2.6 Traditional Applications

We have used the label "traditional applications" to refer to the more established combinatorial optimization problems. Although some of the problems falling into the other categories also fall into this category and vice versa. The problems studied in this part of the handbook fall into the following categories: bin packing, packing, facility dispersion and location, traveling salesperson, Steiner tree, scheduling, planning, generalized assignment, and satisfiability. Let us briefly discuss these categories.

One of the fundamental problems in approximations is the bin packing problem. Chapter 32 discusses online and offline algorithms for one-dimensional bin packing. Chapters 33 and 34 discuss variants of the bin packing problem. This include variations that fall into the following type of problems: the number of items packed is maximized while keeping the number of bins fixed; there is a bound on the number of items that can be packed in each bin; dynamic bin packing, where each item has an arrival and departure time; the item sizes are not known, but the ordering of the weights is known; items may be fragmented while packing them into fixed capacity bins, but certain items cannot be assigned to the same bin; bin stretching; variable sized bin packing problem; and the bin covering problem.

Chapter 35 discusses several ways to generalize the bin packing problem to more dimensions. Two- and three-dimensional strip packing, bin packing in dimensions two and higher, vector packing, and several other variations are discussed. Primal-dual approximation algorithms for packing and stabbing (or covering) problems are covered in Chapter 37. Cutting and packing problems with important applications in the wood, glass, steel, and leather industries as well as in very large-scale integration (VLSI) design, newspaper paging, and container and truck loading are discussed in Chapter 36. For several decades, cutting and packing has attracted the attention of researchers in various areas including operations research, computer science, manufacturing, etc.

Facility dispersion problems are covered in Chapter 38. Dispersion problems arise in a number of applications, such as locating obnoxious facilities, choosing sites for business franchises, and selecting dissimilar solutions in multiobjective optimization. The facility location problem that model the placement of "desirable" facilities such as warehouses, hospitals, and fire stations are discussed in Chapter 39. These algorithms are called "dual fitting and factor revealing."

Very interesting approximation algorithms for the prize collecting TSP is studied in Chapter 40. In this problem a salesperson has to collect a certain amount of prizes (the quota) by visiting cities. A known prize can be collected in every city. Chapter 41 discusses branch-and-bound algorithms for the TSP. These algorithms have been implemented to run in a multicomputer environment. A general software tool for running branch and bound algorithms in a distributed environment is discussed. This framework may be used for almost any divide-and-conquer computation. With minor adjustments, this tool can take any algorithm defined as a computation over directed acyclic graph, where the nodes refer to computations and the edges specify a precedence relation between computations, and run in a distributed environment.

Approximation algorithms for the Steiner tree problem are discussed in Chapter 42. This problem has applications in several research areas. One of these areas is VLSI physical design. In Chapter 43, practical approximations for a restricted Steiner tree problem are discussed.

Meeting deadline constraints is of great importance in real-time systems. In situations when this is not possible, it is often more desirable to execute some parts of every task, than to give up completely the execution of some tasks. This model allows for the trade-off of the quality of computations in favor of meeting the deadline constraints. Every task is logically decomposed into two subtasks, mandatory and optional. This type of scheduling problems fall under the imprecise computation model. These problems are discussed in Chapter 44. Chapter 45 discussed approximation algorithms for the *malleable task* scheduling problem. In this model, the processing time of a task depends on the number of processors allotted to it. A generalization of both the bin packing and TSP is the vehicle scheduling problem. Approximation algorithms for this problem are discussed in Chapter 46.

Automated planning consists of finding a sequence of actions that transforms an initial state into one of the goal states. Planning is widely applicable, and has been used in such diverse application domains as spacecraft control, planetary rover operations, automated nursing aides, image processing, computer security, and automated manufacturing. Chapter 47 discusses approximation algorithms and heuristics for problems falling into this category.

Chapter 48 presents heuristics and metaheuristics for the generalized assignment problem. This problem is a natural generalization of combinatorial optimization problems including bipartite matching, knapsack and bin packing problems; and has many important applications in flexible manufacturing systems, facility location, and vehicle routing problems.

Chapter 49 examines probabilistic greedy heuristics for maximization and minimization versions of the satisfiability problem.

2.7 Computational Geometry and Graph Applications

The problems falling into this category have applications in several fields of study, but can be viewed as computational geometry and graph problems. The problems studied in this part of the handbook fall into the following categories: 2D and 3D triangulations, connectivity problems, design and evaluation of geometric networks, pair decompositions, minimum edge length partitions, digital geometry, disjoint path problems, graph partitioning, graph coloring, finding subgraphs or trees with certain properties, etc.

Triangulation is not only an interesting theoretical problem in computational geometry, it also has many important applications, such as finite element methods for computer-aided design (CAD) and physical simulations. Chapter 50 discusses approximation algorithms for triangulations in two and three dimensions.

Chapter 51 examines approximation schemes for various geometric minimum-cost k-connectivity problems and for geometric survivability problems, giving a detailed tutorial of the novel techniques developed for these algorithms.

Geometric networks arise in many applications. Road networks, railway networks, telecommunication, pattern matching, bioinformatics—any collection of objects in space that have some connections between them can be modeled as a geometric network. Chapter 52 considers the problem of designing a "good" network and the dual problem, i.e., evaluating how "good" a given network is. Chapter 53 gives an overview of several proximity problems that can be solved efficiently using the well-separated pair decomposition (WSPD). A WSPD may be regarded as a "small" set of edges that approximates the dense complete Euclidean graph.

Approximation algorithms for minimum edge length partitions of rectangles with interior points are discussed in Chapter 54. This problem has applications in the area of CAD of integrated circuits and systems. Chapter 55 considers partitions of finite d-dimensional integer grids by lines in two-dimensional space or by hyperplanes and hypersurfaces in an arbitrary dimension. Some of these problems arise in the areas of digital image processing (analysis) and neural networks. Chapter 56 discusses the problem of finding a planar subgraph of maximum weight in a given graph. Problems of this form have applications in circuit layout, facility layout, and graph drawing.

Finding disjoint paths in graphs is a problem that has attracted considerable attention from at least three perspectives: graph theory, VLSI design, and network routing/flow. The corresponding literature is extensive. Chapter 57 explores offline approximation algorithms for problems on general graphs as influenced from the network flow perspective.

Chapter 58 surveys approximation algorithms and hardness results for different versions of the generalized Steiner network problem in which we seek to find a low-cost subgraph that satisfies prescribed connectivity requirements. These problems include the following well-known problems: min-cost k-flow, min-cost spanning tree, traveling salesman, directed/undirected Steiner tree, Steiner forest, k-edge/node-connected spanning subgraph, and others.

Besides numerous network design applications, spanning trees also play an important role in several newly established research areas, such as biological sequence alignments and evolutionary tree construction. Chapter 59 explores the problem of designing approximation algorithms for spanning-tree problems under different objective functions. It focuses on approximation algorithms for constructing efficient communication spanning trees.

Graph partitioning problem arises in a wide range of applications. Due to the complexity of the problem, heuristics have to be applied to partition large graphs in a reasonable amount of time. Chapter 60 discusses different approaches to the graph partitioning problem. The k-way partitioning of a hypergraph problem seeks to minimize a given cost function of such an assignment. A standard cost function is *net cut*, which is the number of hyperedges that span more than one partition, or, more generally, the sum of weight of such edges. Constraints are typically imposed on the solution, and make the problem difficult. Several heuristics for this problem are discussed in Chapter 61.

In many applications such as design of transportation networks, one often needs to identify a set of regions/sections whose damage will cause the greatest increase in transportation cost within the network. Once identified, extra protection can be deployed to prevent them from being damaged. A version of this problem is finding the most vital edges whose removal will cause the greatest damage to a particular property of the graph. The problems are traditionally referred to as prior analysis problems in sensitivity analysis and it is discussed in Chapter 62.

Stochastic local search algorithms for the classical graph coloring problem are discussed in Chapter 63. This problem arises in many real-life applications like register allocation, air traffic flow management, frequency assignment, light wavelengths assignment in optical networks, or timetabling. Chapter 64 discusses ant colony optimization (ACO) for solving the maximum disjoint paths problems. This problem has many applications including the establishment of routes for connection requests between physically separated network endpoints.

2.8 Large-Scale and Emerging Applications

The problems arising in the areas of wireless and sensor networks, multicasting, multimedia, bioinformatics VLSI CAD, game theory, data analysis, digital reputation, and color quantization may be referred to as problems in "emerging" applications and normally involve large-scale problems instances. Some of these problems also fall in the other application areas.

Chapter 65 describes existing multicast routing protocols for ad hoc and sensor networks, and analyze the issue of computing minimum cost multicast trees. The multicast routing problem, and approximation algorithms for mobile ad hoc networks (MANETs) and wireless sensor networks (WSNs) are presented.

Since flat networks do not scale, it is important to overlay a virtual infrastructure on a physical network. The design of the virtual infrastructure should be general enough so that it can be leveraged by a multitude of different protocols. Chapter 66 proposes a novel clustering scheme based on a number of properties of diameter-2 graphs. Extensive simulation results have shown the effectiveness of the clustering scheme when compared to other schemes proposed in the literature.

Ad hoc networks are formed by collections of nodes which communicate with each other through radio propagation. Topology control problems in such networks deal with the assignment of power values to the nodes so that the power assignment leads to a graph topology satisfying some specified properties. The problem is to minimize a specified function of the powers assigned to the nodes. Chapter 67 discusses some known approximation algorithms for this type of problems. The focus is on approximation algorithms with proven performance guarantees.

An important requirement of wireless ad hoc networks is that they should be self-organizing. Energy conservation and network performance are probably the most critical issues in wireless ad hoc networks, because wireless devices are usually powered by batteries only and have limited computing capability and memory. Many proposed methods apply computational geometry technique (specifically, geometrical spanner) to achieve power efficiency. In Chapter 68, approximation algorithms of power spanner for ad hoc networks are reviewed.

As networks continue to grow explosively both in size and internal complexity, the ever-increasing tremendous traffic load and applications drive researchers to develop techniques for analyzing network performance and managing network resources. To accomplish this, one needs to know the current internal structure of the network. Discovery of internal information such as topology and localized lossy links plays an important role in resource management, loss recovery, and congestion control. Chapter 69 proposes a way to identify this via message multicasting.

Due to the recently rapid development of multimedia applications, multicast has become the critical technique in many network applications. In multicasting routing, the main objective is to send data from one or more sources to multiple destinations to minimize the usage of resources such as bandwidth, communication time, and connection costs. Chapter 70 discusses contemporary research concerning multicast congestion problems in different type of networks.

Recent progress in audio, video, and data storage technologies has given rise to a host of high-bandwidth real-time applications such as video conferencing. These applications require Quality of Service (QoS) guarantees from the underlying networks. Thus, multicast routing algorithms, which manage network resources efficiently and satisfy the QoS requirements, have come under increased scrutiny in recent years. Chapter 71 considers the problem of finding an optimal multicast tree with certain special characteristics. This problem is a generalization of the classical Steiner tree problem.

Scalability is especially critical for peer-to-peer systems. The basic idea of peer-to-peer systems is to have an open self-organizing system of peers that does not rely on any central server and where peers can join and leave, at will. This has the benefit that individuals can cooperate without fees or an investment in additional high-performance hardware. Also, peer-to-peer systems can make use of the tremendous amount of resources (such as computation and storage) that otherwise sit idle on individual computers when they are not in use by their owners. Chapter 72 seeks ways of implementing join, leave, and route operations so that for any sequence of join, leave, and route requests can be executed quickly; the degree, diameter, and stretch factor of the resulting network are as small as possible; and the *expansion* of the resulting network is as large as possible. Good approximate solutions to this multiobjective optimization problem are discussed in Chapter 72.

Scheduling problems modeling the broadcasting of data items over wireless channels are discussed in Chapter 73. The chapter covers exact and heuristic solutions for variants of this problem.

Microarrays have been evolving rapidly, and are among the most novel and revolutionary new biotechnologies. They allow us to monitor the expression of thousands of genes at once. With a single experiment billions of individual hypotheses can be tested. Chapter 74 presents three illustrative examples in the analysis of microarray data sets.

Chapter 75 considers two problems from computational biology, namely, primer selection and planted motif search. The closest string and the closest substring problems are closely related to the planted motif search problem. Representative approximation algorithms for these problems are discussed.

There are interesting algorithmic issues that arise when length constraints are taken into account in the formulation of a variety of problems on string similarity, particularly in the problems related to local alignment. Chapter 76 discusses these types of problems which have their roots and most striking applications in computational biology. Chapter 77 discusses approximation algorithms for the selection of robust tag single nucleotide polymorphisms (SNPs). This is a problem in human genomics that arises in the current experimental environment. Chapter 78 considers a sphere packing problem. Recent interest on this problem was motivated by medical applications in radiosurgery. Radiosurgery is a minimally invasive surgical procedure that uses radiation to destroy tumors inside the human body.

VLSI has produced some of the largest combinatorial optimization problems ever considered. Placement is one of the most difficult of these problems. Placement problems with over 10 million variables and constraints are not unusual, and problem sizes continue to grow with Moore's law. Realistic objectives and constraints for placement incorporate complex models of signal timing, power consumption, wiring routability, manufacturability, noise, temperature, etc. Chapter 79 considers VLSI placement algorithms.

Due to delay scaling effects in deep-submicron technologies, interconnect planning and synthesis are becoming critical to meeting VLSI chip performance targets with reduced design turnaround time. In particular, the global routing phase of the design cycle is receiving renewed interest, as it must efficiently handle increasingly more complex constraints for increasingly larger designs. Chapter 80 presents an integrated approach for congestion and timing-driven global routing, buffer insertion, pin assignment, and buffer/wire sizing. This is a multiobjective optimization problem.

Chapters 81–83 discuss game theory problems related to the Internet and scheduling. They deal with ways of achieving equilibrium. Issues related to algorithmic game theory, approximate economic equilibrium and algorithm mechanism design are discussed.

Over the last decade, the size of data seen by a computational problem has grown immensely. There appears to be more web pages than human beings, and web pages have been successfully indexed. Routers generate huge traffic logs, in the order of terabytes, in a short time. The same explosion of data is felt in observational sciences because our capabilities of measurement have grown significantly. Chapter 84 considers a processing mode where input items are not explicitly stored and the algorithm just passes over the data once.

A virtual community can be defined as a group of people sharing a common interest or goal who interact over a virtual medium, most commonly the Internet. Virtual communities are characterized by an absence of face-to-face interaction between participants which makes the task of measuring the trust-worthiness of other participants harder than in nonvirtual communities. This is because of the anonymity that the Internet provides, coupled with the loss of audiovisual cues that help in the establishment of trust. As a result, digital reputation management systems are an invaluable tool for measuring trust in virtual communities. Chapter 85 discusses various systems which can be used to generate a good solution to this problem.

Chapter 86 considers the problem of approximating "colors." Several algorithmic methodologies are presented and evaluated experimentally. These algorithms include dimension weighted clustering approx-imation algorithms.

References

[1] Graham, R. L., Bounds for certain multiprocessing anomalies, *Bell Syst. Tech. J.,* 45, 1563, 1966.

[2] Graham, R. L., Bounds on multiprocessing timing anomalies, *SIAM J. Appl. Math.,* 17, 263, 1969.

[3] Garey, M. R. and Johnson, D. S., *Computers and Intractability: A Guide to the Theory of NP-Completeness,* W. H. Freeman, San Francisco, 1979.

[4] Coffman, E. G., Jr., Garey, M. R., and Johnson, D. S., An application of bin-packing to multiprocessor scheduling, *SIAM J. Comput.,* 7, 1, 1978.

[5] Friesen, D. K., Tighter bounds for the multifit processor scheduling algorithm, *SIAM J. Comput.,* 13, 170, 1984.
[6] Friesen, D. K. and Langston, M. A., Bounds for multifit scheduling on uniform processors, *SIAM J. Comput.,* 12, 60, 1983.
[7] Bar-Yehuda, R. and Even, S., A linear time approximation algorithm for the weighted vertex cover problem, *J. Algorithms*, 2, 198, 1981.
[8] Gonzalez, T. F., A simple LP-free approximation algorithm for the minimum web pages vertex cover problem, *Inform. Proc. Lett.*, 54(3), 129, 1995.
[9] Bar-Yehuda, R. and Even, S., A local-ratio theorem for approximating the weighted set cover problem, *Ann. Disc. Math.*, 25, 27, 1985.
[10] Bar-Yehuda, R. and Bendel, K., Local ratio: a unified framework for approximation algorithms, *ACM Comput. Surv.*, 36(4), 422, 2004.
[11] Schrijver, A., *Theory of Linear and Integer Programming*, Wiley-Interscience Series in Discrete Mathematics and Optimization, Wiley, New York, 2000.
[12] Vanderbei, R. J., *Linear Programming Foundations and Extensions,* International Series in Operations Research & Management Science, Vol. 37, Springer, Berlin, 2001.
[13] Sahni, S. and Gonzalez, T., P-complete approximation problems, *JACM*, 23, 555, 1976.