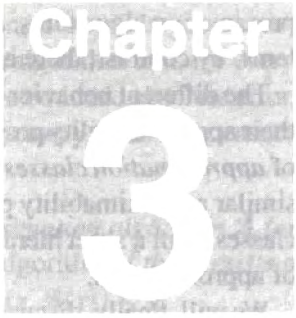


Approximation Classes

In the first chapter we have seen that, due to their inherent complexity, NP-hard optimization problems cannot be efficiently solved in an exact way, unless P = NP. Therefore, if we want to solve an NP-hard optimization problem by means of an efficient (polynomial-time) algorithm, we have to accept the fact that the algorithm does not always return an optimal solution but rather an approximate one. In Chap. 2, we have seen that, in some cases, standard algorithm design techniques, such as local search or greedy techniques, although inadequate to obtain the optimal solution of NP-hard optimization problems, are powerful enough to reach good approximate solutions in polynomial time.

In this chapter and in the following one, we will formally introduce an important type of approximation algorithms. Given an instance of an NP-hard optimization problem, such algorithms provide a solution whose performance ratio is guaranteed to be bounded by a suitable function of the input size. This type of approximation is usually called *performance guarantee approximation*. In particular, while in the next chapter we will deal with slowly increasing bounding functions, we will here consider the case in which the function is a constant. An example of this kind is the greedy algorithm for MAXIMUM KNAPSACK, which we have already met in Sect. 2.1, that efficiently finds a solution whose value is always at least one half of the optimal one.

After giving the basic definitions related to the notion of performance guarantee approximation, we will state both positive results, showing that several computationally hard problems allow efficient, arbitrarily good ap-

proximation algorithms, and negative results, showing that for other problems, by contrast, there are intrinsic limitations to approximability.

The different behavior of NP-hard optimization problems with respect to their approximability properties will be captured by means of the definition of *approximation classes*, that is, classes of optimization problems sharing similar approximability properties. We will see that, if $P \neq NP$, then these classes form a strict hierarchy whose levels correspond to different degrees of approximation.

We will, finally, discuss some conditions under which optimization problems allow approximation algorithms with arbitrarily good guaranteed performance.

3.1 Approximate solutions with guaranteed performance

BEFORE WE formally introduce algorithms that provide approximate solutions of optimization problems with guaranteed quality, let us first observe that according to the general framework set up in Chap. 1, given an input instance x of an optimization problem, any feasible solution $y \in \mathrm{SOL}(x)$ is indeed an approximate solution of the given problem. On such a ground an approximation algorithm may be defined as follows.

Definition 3.1 ▶ *Approximation algorithm*

Given an optimization problem $\mathcal{P} = (I, \mathrm{SOL}, m, \mathrm{goal})$, an algorithm $\mathcal{A}$ is an approximation algorithm *for $\mathcal{P}$ if, for any given instance $x \in I$, it returns an* approximate solution*, that is a feasible solution $\mathcal{A}(x) \in \mathrm{SOL}(x)$.*

3.1.1 Absolute approximation

Clearly, for all practical purposes, Def. 3.1 is unsatisfactory. What we are ready to accept as an approximate solution is a feasible solution whose value is 'not too far' from the optimum. Therefore we are interested in determining how far the value of the achieved solution is from the value of an optimal one.

Definition 3.2 ▶ *Absolute error*

Given an optimization problem $\mathcal{P}$, for any instance x and for any feasible solution y of x, the absolute error *of y with respect to x is defined as*

$$D(x,y) = |m^*(x) - m(x,y)|$$

where $m^*(x)$ denotes the measure of an optimal solution of instance x and $m(x,y)$ denotes the measure of solution y.

Indeed, given an NP-hard optimization problem, it would be very satisfactory to have a polynomial-time approximation algorithm that, for every instance x, is capable of providing a solution with a bounded absolute error, that is a solution whose measure is only a constant away from the measure of an optimal one.

◀ Definition 3.3
Absolute approximation algorithm

Given an optimization problem $\mathcal{P}$ and an approximation algorithm $\mathcal{A}$ for $\mathcal{P}$, we say that $\mathcal{A}$ is an absolute approximation algorithm *if there exists a constant k such that, for every instance x of $\mathcal{P}$, $D(x, \mathcal{A}(x)) \leq k$.*

◀ Example 3.1

Let us consider the problem of determining the minimum number of colors needed to color a planar graph. We have seen in Theorem 2.12 that a 6-coloring of a planar graph can be found in polynomial time. It is also known that establishing whether the graph is 1-colorable (that is, the set of edges is empty) or 2-colorable (that is, the graph is bipartite) is decidable in polynomial time whereas the problem of deciding whether three colors are enough is NP-complete. Clearly, if we provide an algorithm that returns either a 1- or a 2-coloring of the nodes if the graph is 1- or 2-colorable, respectively, and returns a 6-coloring in all other cases, we obtain an approximate solution with absolute error bounded by 3.

A second related (but less trivial) example, concerning the edge coloring problem, will be considered in Chap. 4.

Unfortunately, there are few cases in which we can build absolute approximation algorithms and, in general, we cannot expect such a good performance from an approximation algorithm. The knapsack problem is an example of an NP-hard problem that does not allow a polynomial-time approximation algorithm with bounded absolute error.

◀ Theorem 3.1

Unless P = NP, *no polynomial-time absolute approximation algorithm exists for* MAXIMUM KNAPSACK.

PROOF

Let X be a set of n items with profits $p_1, \ldots, p_n$ and sizes $a_1, \ldots, a_n$, and let b be the knapsack capacity. If the problem would allow a polynomial-time approximation algorithm with absolute error k, then we could solve the given instance exactly in the following way. Let us create a new instance by multiplying all profits p_i by $k+1$. Clearly, the set of feasible solutions of the new instance is the same as that of the original instance. On the other side, since the measure of any solution is now a multiple of $k+1$, the only solution with absolute error bounded by k is the optimal solution. Hence, if we knew how to find such a solution in polynomial time, we would also be able to exactly solve the original instance in polynomial time.

QED

Similar arguments apply to show that most other problems, such as MINIMUM TRAVELING SALESPERSON and MAXIMUM INDEPENDENT SET, do not allow polynomial-time absolute approximation algorithms.

3.1.2 Relative approximation

In order to express the quality of an approximate solution, more interesting and more commonly used notions are the relative error and the performance ratio.

Definition 3.4 ▶ *Relative error*

Given an optimization problem $\mathcal{P}$, for any instance x of $\mathcal{P}$ and for any feasible solution y of x, the relative error *of y with respect to x is defined as*

$$E(x,y) = \frac{|m^*(x) - m(x,y)|}{\max\{m^*(x), m(x,y)\}}.$$

Both in the case of maximization problems and of minimization problems, the relative error is equal to 0 when the solution obtained is optimal, and becomes close to 1 when the approximate solution is very poor.

Definition 3.5 ▶ *ε-approximate algorithm*

Given an optimization problem $\mathcal{P}$ and an approximation algorithm $\mathcal{A}$ for $\mathcal{P}$, we say that $\mathcal{A}$ is an ε-approximate algorithm *for $\mathcal{P}$ if, given any input instance x of $\mathcal{P}$, the relative error of the approximate solution $\mathcal{A}(x)$ provided by algorithm $\mathcal{A}$ is bounded by ε, that is:*

$$E(x, \mathcal{A}(x)) \leq \varepsilon.$$

The greedy algorithm we analyzed in Sect. 2.1 for MAXIMUM KNAPSACK is an example of a polynomial-time $1/2$-approximate algorithm. In fact, such an algorithm always provides a solution whose relative error is at most $1/2$.

As an alternative to the relative error, we can express the quality of the approximation by means of a different, but related, measure.

Definition 3.6 ▶ *Performance ratio*

Given an optimization problem $\mathcal{P}$, for any instance x of $\mathcal{P}$ and for any feasible solution y of x, the performance ratio *of y with respect to x defined as*

$$R(x,y) = \max\left(\frac{m(x,y)}{m^*(x)}, \frac{m^*(x)}{m(x,y)}\right).$$

Both in the case of minimization problems and of maximization problems, the value of the performance ratio is equal to 1 in the case of an optimal solution, and can assume arbitrarily large values in the case of poor approximate solutions. The fact of expressing the quality of approximate solutions by a number larger than 1 in both cases is slightly counterintuitive, but it yields the undoubted advantage of allowing a uniform treatment of minimization and maximization problems.

Clearly the performance ratio and the relative error are related. In fact, in any case, the relative error of solution y on input x is equal to $E(x,y) = 1 - 1/R(x,y)$.

As in the case of the relative error, also for the performance ratio it is interesting to consider situations in which such a quality measure is bounded by a constant for all input instances.

◀ Definition 3.7
r-approximate algorithm

Given an optimization problem $\mathcal{P}$ and an approximation algorithm $\mathcal{A}$ for $\mathcal{P}$, we say that $\mathcal{A}$ is an r-approximate algorithm *for $\mathcal{P}$ if, given any input instance x of $\mathcal{P}$, the performance ratio of the approximate solution $\mathcal{A}(x)$ is bounded by r, that is:*

$$R(x, \mathcal{A}(x)) \leq r.$$

For example, the greedy algorithm for MAXIMUM KNAPSACK is an example of a 2-approximate algorithm since it always provides a solution whose value is at least one half of the optimal value.

Notice that, according to our definition, if, for a given optimization problem $\mathcal{P}$ and a given algorithm $\mathcal{A}$ for $\mathcal{P}$, we have that, for all instances x of $\mathcal{P}$, $m_{\mathcal{A}}(x,y) \leq rm^*(x) + k$ we do not say that algorithm $\mathcal{A}$ is r-approximate, but that it is at most $r+k$-approximate. In the literature it is widely accepted that, in such case, the algorithm is called r-approximate, under an asymptotic point of view. For example, Theorem 2.9 states that, given an instance x of MINIMUM BIN PACKING, *First Fit Decreasing* returns a solution such that $m_{FFD}(x) \leq \frac{3}{2}m^*(x) + 1$. Such an algorithm is indeed usually known as a $\frac{3}{2}$-approximate algorithm. The asymptotic point of view in the evaluation of the performance ratio of algorithms will be taken into consideration in Chap. 4.

As we have seen, in order to qualify the degree of approximation achieved by an approximation algorithm we may refer either to the bound ε on its relative error (and speak of an ε-approximate algorithm) or to the bound r on its performance ratio (and speak of an r-approximate algorithm). In the following, we will mainly refer to the performance ratio in order to estimate the quality of the computed solutions: however, since the relative error is always smaller than 1 and the performance ratio is always larger than or equal to 1, it will be always clear which approximation quality measure we are referring to.

The existence of polynomial-time approximation algorithms qualifies different kinds of NP-hard optimization problems.

◀ Definition 3.8
ε-approximable problem

An NP*-hard optimization problem $\mathcal{P}$ is* ε-approximable *(respectively,* r-approximable*) if there exists a polynomial-time* ε*-approximate (respectively,* r*-approximate) algorithm for $\mathcal{P}$.*

Program 3.1: Greedy Sat

```
input Set C of disjunctive clauses on a set of variables V;
output Truth assignment f : V ↦ {TRUE, FALSE};
begin
  for all v ∈ V do f(v) := TRUE;
  repeat
    Let l be the literal that occurs in the maximum number of
      clauses in C (solve ties arbitrarily);
    Let C_l be the set of clauses in which l occurs;
    Let C_l̄ be the set of clauses in which l̄ occurs;
    Let v_l be the variable corresponding to l;
    if l is positive then
    begin
      C := C − C_l;
      Delete l̄ from all clauses in C_l̄; Delete all empty clauses in C
    end
    else
    begin
      f(v_l) := FALSE; C := C − C_l̄;
      Delete l from all clauses in C_l; Delete all empty clauses in C
    end
  until C = ∅;
  return f
end.
```

MAXIMUM KNAPSACK is an example of a 2-approximable problem. In the previous chapter, we have seen several other examples of approximable problems, and more examples will be shown in this chapter and in the following ones.

Let us now consider MAXIMUM SATISFIABILITY. For this problem, we have already shown in Sect. 2.6 a randomized algorithm whose expected performance ratio is bounded by 2. We now show a deterministic 2-approximate algorithm for MAXIMUM SATISFIABILITY which runs in polynomial time (namely, Program 3.1), that is a simple example of applying the greedy technique and can be considered as a "derandomized" version of the algorithm shown in the previous chapter (a different approach based on the local search technique can also be followed in order to obtain a similar result, as stated in Exercise 3.1).

Theorem 3.2 ▶ *Program 3.1 is a polynomial-time 2-approximate algorithm for* MAXIMUM SATISFIABILITY.

PROOF Given an instance with c clauses, we prove, by induction on the number of

variables, that Program 3.1 always satisfies at least $c/2$ clauses. Since no optimal solution can have value larger than c, the theorem will follow.

The result is trivially true in the case of one variable. Let us assume that it is true in the case of $n-1$ variables ($n > 1$) and let us consider the case in which we have n variables. Let v be the variable corresponding to the literal which appears in the maximum number of clauses. Let c_1 be the number of clauses in which v appears positive and c_2 be the number of clauses in which v appears negative. Without loss of generality suppose that $c_1 \geq c_2$, so that the algorithm assigns the value TRUE to v. After this assignment, at least $c - c_1 - c_2$ clauses, on $n-1$ variables, must be still considered. By inductive hypothesis, Program 3.1 satisfies at least $(c-c_1-c_2)/2$ such clauses. Hence, the overall number of satisfied clauses is at least $c_1 + (c-c_1-c_2)/2 \geq c/2$.

QED

Within the class NPO, the class of problems that allow polynomial-time r-approximate algorithms (or, equivalently, ε-approximate algorithms) plays a very important role. In fact, the existence of a polynomial-time r-approximate algorithm for an NP-hard optimization problem shows that, despite the inherent complexity of finding the exact solution of the problem, such a solution can somehow be approached.

◀ Definition 3.9
Class APX

APX *is the class of all* NPO *problems* $\mathcal{P}$ *such that, for some* $r \geq 1$, *there exists a polynomial-time r-approximate algorithm for* $\mathcal{P}$.

As shown above and in the previous chapter, MAXIMUM SATISFIABILITY, MAXIMUM KNAPSACK, MAXIMUM CUT, MINIMUM BIN PACKING, MINIMUM GRAPH COLORING restricted to planar graphs, MINIMUM SCHEDULING ON IDENTICAL MACHINES, and MINIMUM VERTEX COVER are all in APX.

The definition of the class APX provides a first important notion for characterizing NPO problems with respect to their degree of approximability. For several important NPO problems, in fact, it can be shown that they do not allow any r-approximate algorithm, unless P = NP. In other words, for these problems, approximate solutions with guaranteed performance are as hard to determine as the optimal solutions. This means that, under the hypothesis that P $\neq$ NP, the class APX is strictly contained in the class NPO.

In the next subsection we will show that MINIMUM TRAVELING SALESPERSON is a problem for which determining approximate solutions with constant bounded performance ratio is computationally intractable. Other NPO problems that do not belong to APX, such as MAXIMUM CLIQUE and MAXIMUM INDEPENDENT SET, will be seen in Chap. 6, where some techniques needed to prove such results will also be provided.

3.1.3 Approximability and non-approximability of TSP

MINIMUM TRAVELING SALESPERSON is an important example of an optimization problem that cannot be r-approximated, no matter how large is the performance ratio r.

Theorem 3.3 ▶ *If* MINIMUM TRAVELING SALESPERSON *belongs to* APX, *then* P = NP.

PROOF The result is proved by reduction from the HAMILTONIAN CIRCUIT decision problem. HAMILTONIAN CIRCUIT is the problem of deciding whether a directed graph admits a circuit which passes exactly once through every node: this problem is known to be NP-complete. Let $G = (V,E)$ be an instance of HAMILTONIAN CIRCUIT with $|V| = n$. For any $r \geq 1$, we construct an instance of MINIMUM TRAVELING SALESPERSON such that if we had a polynomial-time r-approximate algorithm for MINIMUM TRAVELING SALESPERSON, then we could decide whether the graph G has a Hamiltonian circuit in polynomial time. From this construction the theorem will then follow.

Given $G = (V,E)$, the instance of MINIMUM TRAVELING SALESPERSON is defined on the same set of nodes V and with distances

$$d(v_i, v_j) = \begin{cases} 1 & \text{if } (v_i, v_j) \in E, \\ 1 + nr & \text{otherwise.} \end{cases}$$

This instance of MINIMUM TRAVELING SALESPERSON has a solution of measure n if and only if the graph G has a Hamiltonian circuit: in that case, the next smallest approximate solution has measure at least $n(1+r)$ and its performance ratio is hence greater than r. Besides, if G has no Hamiltonian circuit, then the optimal solution has measure at least $n(1+r)$. Therefore, if we had a polynomial-time r-approximate algorithm for MINIMUM TRAVELING SALESPERSON, we could use it to decide whether G has a Hamiltonian circuit in the following way: we apply the approximation algorithm to the instance of MINIMUM TRAVELING SALESPERSON corresponding to G and we answer YES if and only if it returns a solution of measure n.

QED

Corollary 3.4 ▶ *If* P ≠ NP, *then* APX ⊂ NPO.

The hardness of approximating MINIMUM TRAVELING SALESPERSON significantly reduces if we make suitable assumptions on the problem instances. In particular, recall that MINIMUM METRIC TRAVELING SALESPERSON is defined as MINIMUM TRAVELING SALESPERSON restricted to instances that satisfy the triangular inequality (see Sect. 2.1.3). Note also

that, as a particular case, all instances of MINIMUM TRAVELING SALESPERSON which make use of the Euclidean distance satisfy the triangular inequality.

From a complexity point of view, MINIMUM TRAVELING SALESPERSON does not really become easier when the triangular inequality is satisfied: indeed, it remains NP-hard. From an approximation point of view the situation is quite different and MINIMUM METRIC TRAVELING SALESPERSON can be shown to be, in a sense, "easier" than MINIMUM TRAVELING SALESPERSON. In fact, we now define a 3/2-approximate polynomial-time algorithm for MINIMUM METRIC TRAVELING SALESPERSON, called *Christofides' algorithm*.

To this aim, let us first introduce some definitions and preliminary results that are needed for understanding and analyzing the algorithm.

A *multigraph* is a pair $M = (V, F)$ where V is a finite set of nodes and F is a multiset of edges. A *weighted multigraph* is a multigraph where a weight $c(e)$ is associated to each edge $e \in F$. A *walk* on a multigraph is a sequence of nodes $(v_1, \ldots, v_m)$, where each node may appear more than once and such that, for every i with $1 \leq i < m$, there is an edge connecting v_i and v_{i+1}. A walk $(v_1, \ldots, v_m)$ is said to be *closed* if $v_1 = v_m$. An *Eulerian walk* is a closed walk in which each node is visited at least once and each edge is traversed exactly once. A multigraph is *Eulerian* if it admits an Eulerian walk. For example, the multigraph of Fig. 3.1 is an Eulerian graph since the walk

$$(v_1, v_2, v_3, v_5, v_6, v_4, v_3, v_2, v_1, v_6, v_5, v_4, v_1)$$

is Eulerian.

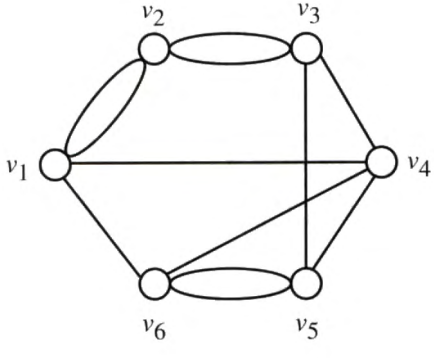


Figure 3.1
An Eulerian multigraph

It is well-known that a polynomial-time algorithm exists that, given in input a multigraph M, decides whether M is Eulerian and, eventually, returns an Eulerian walk on M (see Bibliographical notes). Christofides'

algorithm is based on this fact and on the possibility of extracting a tour from an Eulerian walk, as stated in the following lemma.

Lemma 3.5 ▶ *Let $G = (V, E)$ be a complete weighted graph satisfying the triangular inequality and let $M = (V, F)$ be any Eulerian multigraph over the same set of nodes such that all edges between two nodes u and v in M have the same weight as the edges (u, v) in G. Then, we can find in polynomial time a tour I in G whose measure is at most equal to the sum of the weights of the edges in M.*

PROOF Let w be any Eulerian walk over M. Since all nodes $v_1, \ldots, v_n$ appear at least once in the walk, there must exist at least one permutation $\pi(1), \ldots, \pi(n)$ of the integers in $\{1, \ldots, n\}$ such that w can be written as $(v_{\pi(1)}, \alpha_1, v_{\pi(2)}, \alpha_2, \ldots, v_{\pi(n)}, \alpha_n, v_{\pi(1)})$ where the symbols $\alpha_1, \ldots, \alpha_n$ denote (possibly empty) sequences of nodes (for example, such a permutation can be obtained by considering the first occurrences of all nodes). Due to the triangular inequality, the weight of any edge $(v_{\pi(j)}, v_{\pi(j+1)})$, with $1 \leq j < n$, is no greater than the total weight of the path $(v_{\pi(j)}, \alpha_j, v_{\pi(j+1)})$ and the weight of the edge $(v_{\pi(n)}, v_{\pi(1)})$ is no greater than the total weight of the path $(v_{\pi(n)}, \alpha_n, v_{\pi(1)})$. Hence, if we consider the tour I corresponding to permutation π, the measure of I is at most equal to the total weight of the Eulerian walk w, that is, the sum of the weights of the edges in M. QED

Given an instance G of MINIMUM METRIC TRAVELING SALESPERSON, a general approach to approximately solve this instance could then consists of the following steps: (1) compute a spanning tree T of G in order to visit all nodes, (2) starting from T, derive a multigraph M satisfying the hypothesis of the previous lemma, (3) compute an Eulerian walk w on M, and (4) extract from w a tour according to the proof of the lemma.

The only unspecified step of this approach is how the multigraph M is derived from T. A simple way to perform this step consists of just doubling all edges in T: that is, for each edge (u, v) in T, M contains two copies of this edge with the same weight. It can be easily shown that such an algorithm returns a tour whose performance ratio is bounded by 2 (see Exercise 3.5).

A more sophisticated application of this approach is shown in Program 3.2 where the multigraph M is obtained by adding to T the edges of a minimum weight perfect matching among the vertices in T of odd degree. The following result shows that this new algorithm provides a sensibly better performance ratio.

Theorem 3.6 ▶ *Given an instance G of* MINIMUM METRIC TRAVELING SALESPERSON,

Program 3.2: Christofides

```
input Weighted complete graph G = (V, E);
output Tour I;
begin
    Find a minimum spanning tree T = (V, E_T) in G;
    Let C be the set of nodes in T with odd degree;
    Find a minimum weight perfect matching H in the subgraph of G induced by C;
    Create a multigraph M = (V, E_T ∪ H);
    Find a Eulerian walk w in M;
    Extract a tour I from w (according to the proof of Lemma 3.5);
    return I
end.
```

Christofides' algorithm returns, in polynomial time, a solution of G whose performance ratio is at most 3/2.

PROOF

Let us consider the multigraph $M = (V, E_T \cup H)$ and an Eulerian walk w on M. Let us denote by $c(T)$ and $c(H)$ the sums of the weights of the edges belonging to T and H, respectively. Since M clearly satisfies the hypothesis of Lemma 3.5, we can find in polynomial time a tour I such that

$$m(G,I) \leq c(T) + c(H). \tag{3.1}$$

We now prove the following two facts.

Fact 1: $m^*(G) \geq 2c(H)$. Let $(v_{i_1}, \ldots, v_{i_{2|H|}})$ be the sequence of odd-degree vertices of T in the order in which they appear in an optimal tour I^*. Let H_1 and H_2 be the following two matchings:

$$H_1 = \{(v_1, v_2), (v_3, v_4), \ldots, (v_{k-1}, v_k)\}$$

and

$$H_2 = \{(v_2, v_3), \ldots, (v_k, v_1)\}.$$

By making use of the triangular inequality, we have that $m^*(G) \geq c(H_1) + c(H_2)$, where $c(H_i)$ denotes the sum of the weights of the edges in H_i, for $i = 1, 2$. Since H is a minimum weight perfect matching, we have that both $c(H_1)$ and $c(H_2)$ are greater than or equal to $c(H)$ and, hence, $m^*(G) \geq 2c(H)$.

Fact 2: $c(T) \leq m^*(G)$. In order to prove this fact, it is enough to observe that a tour is also a spanning tree (plus one additional edge): since T is a minimum spanning tree, we have $c(T) \leq m^*(G)$.

From Eq. 3.1 and from the above two facts, we then have that $m(G,I) \leq m^*(G) + m^*(G)/2 = 3m^*(G)/2$.

As far as the running time is concerned, it is easy to see that the most expensive step is the computation of the minimum weight perfect matching. This task can, indeed, be performed in polynomial time by means of a primal-dual algorithm (see Bibliographical notes).

QED

Example 3.2 ▶ Let us consider the instance G of MINIMUM METRIC TRAVELING SALESPERSON consisting of eight cities (i.e., Amsterdam, Berlin, Geneva, Milan, Prague, Rome, Warsaw, and Wien) with distances shown in Fig. 3.2 (these distances are the real road-distances between the specified cities).

AMS	685	925	1180	960	1755	1235	1180
BER		1160	1105	340	1530	585	630
GEN			325	950	880	1575	1025
MIL				870	575	1495	830
PRA					1290	625	290
ROM						1915	1130
WAR							795
WIE							

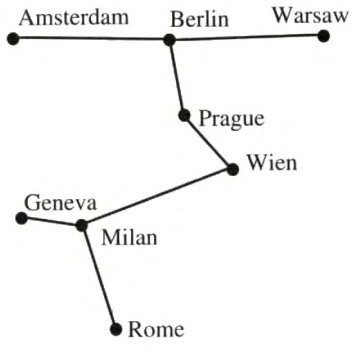
Figure 3.2
An instance of MINIMUM METRIC TRAVELING SALESPERSON

In Fig. 3.3(a) the minimum spanning tree T of G computed in the first step of Christofides' algorithm is shown: note that, in this tree, there are six odd-degree nodes (that is, Amsterdam, Berlin, Geneva, Milan, Rome, and Warsaw). A minimum weight perfect matching H among these six nodes consists of edges Amsterdam-Geneva, Berlin-Warsaw, and Milan-Rome. In Fig. 3.3(b) the multigraph M obtained by joining T and H is given: clearly, this is an Eulerian graph satisfying the hypothesis of Lemma 3.5. An Eulerian walk on M starting from Amsterdam is

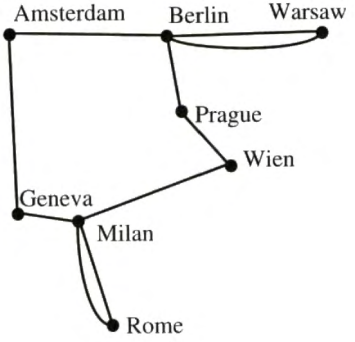
AMS-BER-WAR-BER-PRA-WIE-MIL-ROM-MIL-GEN-AMS.

By considering only the first occurrence of each city in the walk, we obtain the approximate tour shown in Fig 3.3(c) whose measure is 5395. In Fig 3.3(d) the optimal tour is given: the corresponding optimal measure is 5140, that is, about 5% better than the measure of the approximate tour.

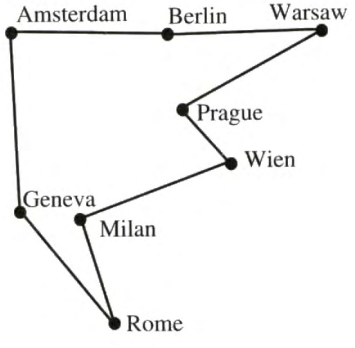
As a consequence of Theorem 3.6, we have that MINIMUM METRIC TRAVELING SALESPERSON belongs to the class APX. The next example shows that the bound of the theorem is tight: that is, there exists a family of weighted complete graphs such that Christofides' algorithm returns a solution whose measure is asymptotically 50% greater than the measure of an optimal tour.



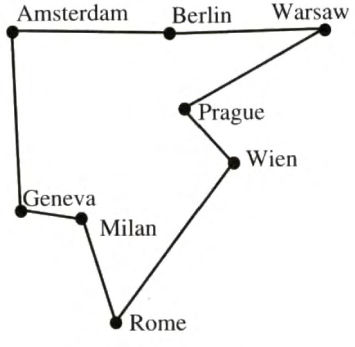
(a) The spanning tree T



(b) The multigraph M



(c) The approximate tour I



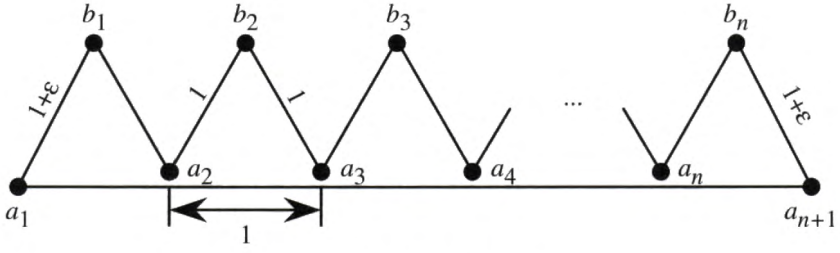
(d) The optimal tour I^*

Figure 3.3
A sample application of Christofides' algorithm

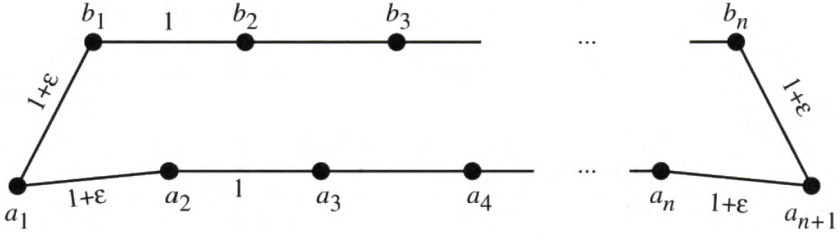
◀ Example 3.3

For any positive integer n, let us consider the instance G_n of MINIMUM METRIC TRAVELING SALESPERSON shown in Fig. 3.4(a), where all distances that are not explicitly specified must be assumed to be computed according to the Euclidean distance. It is easy to see that one possible minimum spanning tree is the tour shown in the figure without the edge (a_1, a_{n+1}). If Christofides' algorithm chooses this spanning tree, then the approximate tour shown in figure results: note that this tour has measure $3n+2\varepsilon$. On the contrary, the optimal tour is shown in Fig. 3.4(b) and has measure $2n+1+4\varepsilon$. As n grows to infinity, the ratio between these two measures approaches the bound $3/2$.

Until now Christofides' algorithm is the best known approximation algorithm for MINIMUM METRIC TRAVELING SALESPERSON. While in the Euclidean case arbitrarily good polynomial-time approximation algorithms can be found (see Bibliographical notes), no polynomial-time approximation algorithm with a better guaranteed performance ratio is known for MINIMUM METRIC TRAVELING SALESPERSON, neither it is known whether the existence of any such algorithm would imply P = NP.



(a) The approximate tour



(b) The optimal tour

Figure 3.4
Worst case of Christofides' algorithm

3.1.4 Limits to approximability: The gap technique

From the point of view of practical applications, it may be observed that knowing that a problem belongs to APX, while interesting in itself, is only partially satisfactory. In fact, in some cases, the existence of guaranteed approximate solutions with large relative errors (e.g., a 50% error as in the case of MAXIMUM SATISFIABILITY or MINIMUM VERTEX COVER) may not be enough for practical purposes. We may be interested in finding stronger approximations, with much smaller relative errors (say 1%). With respect to this need, the optimization problems that belong to APX may behave quite differently. For some problems, not only we can find tight approximate solutions but we can even find arbitrarily good approximate solutions. For such problems we can construct particular polynomial-time algorithms, called polynomial-time approximation schemes (see Sects. 2.5 and 3.2), that, given an instance x of the problem and a constant $r > 1$, produce an r-approximate solution for x. For other problems (unfortunately, the vast majority of problems in APX) the performance ratio can only be reduced up to a certain point: sometimes the approximation techniques can even lead to very tight approximate solutions, but then a threshold t exists such that r-approximability, with $r < t$, becomes computationally intractable.

In order to prove this latter type of result, a simple but powerful technique is frequently used. Such technique is known as *gap technique* and is strictly related to the technique that we have used for proving the non-approximability of MINIMUM TRAVELING SALESPERSON. The technique will now be described in the case of minimization problems but it can also be applied to maximization problems by performing simple modifications to our exposition.

◀ Theorem 3.7

Let $\mathcal{P}'$ be an NP*-complete decision problem and let $\mathcal{P}$ be an* NPO *minimization problem. Let us suppose that there exist two polynomial-time computable function $f : I_{\mathcal{P}'} \mapsto I_{\mathcal{P}}$ and $c : I_{\mathcal{P}'} \mapsto \mathbf{N}$ and a constant* $\mathtt{gap} > 0$*, such that, for any instance x of $\mathcal{P}'$,*

$$m^*(f(x)) = \begin{cases} c(x) & \text{if } x \text{ is a positive instance,} \\ c(x)(1+\mathtt{gap}) & \text{otherwise.} \end{cases}$$

Then no polynomial-time r-approximate algorithm for $\mathcal{P}$ with $r < 1+\mathtt{gap}$ can exist, unless P = NP.

PROOF

Suppose we have a polynomial-time r-approximate algorithm $\mathcal{A}$ with $r < 1+\mathtt{gap}$ for problem $\mathcal{P}$. We can make use of this algorithm for solving problem $\mathcal{P}'$ in polynomial time in the following way. Given an instance x of $\mathcal{P}'$, we compute $f(x)$ and then we apply the approximation algorithm $\mathcal{A}$ to $f(x)$. Let us distinguish the following two cases.

1. x is a negative instance. By hypothesis, in this case $m^*(f(x)) \geq c(x)(1+\mathtt{gap})$ and, *a fortiori*, $m(f(x), \mathcal{A}(x)) \geq c(x)(1+\mathtt{gap})$.
2. x is a positive instance. In this case, since $\mathcal{A}$ is an r-approximate algorithm, we have that

$$\frac{m(f(x), \mathcal{A}(x))}{m^*(f(x))} \leq r < 1+\mathtt{gap}.$$

By hypothesis, $m^*(f(x)) = c(x)$. Hence, $m(f(x), \mathcal{A}(x)) < c(x)(1+\mathtt{gap})$.

Therefore, x is a positive instance of $\mathcal{P}'$ if and only if $m(f(x), \mathcal{A}(x)) < c(x)(1+\mathtt{gap})$, and we would be able to solve problem $\mathcal{P}'$ in polynomial time. Since $\mathcal{P}'$ is NP-complete, this would imply P = NP.

QED

◀ Example 3.4

Let us consider MINIMUM GRAPH COLORING. For this problem, the gap technique can be applied by reduction from the coloring problem for planar graphs. In fact, while a well-known result states that any planar graph is colorable with

at most four colors (see Bibliographical notes), the problem of deciding whether a planar graph is colorable with at most three colors is NP-complete. Hence, in this case, the gap is $1/3$ and we can hence conclude that no polynomial-time r-approximate algorithm for MINIMUM GRAPH COLORING can exist with $r < 4/3$, unless P = NP. Actually, much stronger results hold for the graph coloring problem: it has been proved that, if P $\neq$ NP, then no polynomial-time algorithm can provide an approximate solution whatsoever (that is, MINIMUM GRAPH COLORING belongs to NPO – APX).

The considerations of the previous example can be extended to show that, for any NPO minimization problem $\mathcal{P}$, if there exists a constant k such that it is NP-hard to decide whether, given an instance x, $m^*(x) \leq k$, then no polynomial-time r-approximate algorithm for $\mathcal{P}$ with $r < (k+1)/k$ can exist, unless P = NP (see Exercise 3.8). Another application of the gap technique has been shown in the proof of Theorem 3.3: in that case, actually, we have seen that the constant gap can assume any value greater than 0. Other results which either derive bounds on the performance ratio that can be achieved for particular optimization problems or prove that a problem does not allow a polynomial-time approximation scheme can be obtained by means of a sophisticated use of the gap technique. Such results will be discussed in Chap. 6.

3.2 Polynomial-time approximation schemes

AS WE noticed before, for most practical applications, we need to approach the optimal solution of an optimization problem in a stronger sense than it is allowed by an r-approximate algorithm. Clearly, if the problem is intractable, we have to restrict ourselves to approximate solutions, but we may wish to find better and better approximation algorithms that bring us as close as possible to the optimal solution. Then, in order to obtain r-approximate algorithms with better performances, we may be also ready to pay the cost of a larger computation time, a cost that, as we may expect, will increase with the inverse of the performance ratio.

Definition 3.10 ▶ *Polynomial-time approximation scheme*

Let $\mathcal{P}$ be an NPO *problem. An algorithm $\mathcal{A}$ is said to be a* polynomial-time approximation scheme (PTAS) *for $\mathcal{P}$ if, for any instance x of $\mathcal{P}$ and any rational value $r > 1$, $\mathcal{A}$ when applied to input (x, r) returns an r-approximate solution of x in time polynomial in $|x|$.*

While always being polynomial in $|x|$, the running time of a PTAS may also depend on $1/(r-1)$: the better is the approximation, the larger may be the running time. In most cases, we can indeed approach the optimal solution of a problem arbitrarily well, but at the price of a dramatic increase

Problem 3.1: Minimum Partition

INSTANCE: Finite set X of items, for each $x_i \in X$ a weight $a_i \in Z^+$.

SOLUTION: A partition of the items into two sets Y_1 and Y_2.

MEASURE: $\max\{\sum_{x_i \in Y_1} a_i, \sum_{x_i \in Y_2} a_i\}$.

Program 3.3: Partition PTAS

```
input Set of items X with integer weights a_i, rational r > 1;
output Partition of X into two sets Y_1 and Y_2;
begin
  if r ≥ 2 then return X, ∅
  else
  begin
    Sort items in non-increasing order with respect to their weight;
    (*Let (x_1, ..., x_n) be the obtained sequence*)
    k(r) := ⌈(2 − r)/(r − 1)⌉;
    (* First phase *)
    Find an optimal partition Y_1, Y_2 of x_1, ..., x_k(r);
    (* Second phase *)
    for j := k(r) + 1 to n do
      if Σ_{x_i ∈ Y_1} a_i ≤ Σ_{x_i ∈ Y_2} a_i then
        Y_1 := Y_1 ∪ {x_j}
      else
        Y_2 := Y_2 ∪ {x_j};
    return Y_1, Y_2
  end;
end.
```

in the computation cost. In other cases, we may construct approximation schemes whose running time is polynomial both in the size of the instance and in the inverse of the required degree of approximation. In such cases the possibility of approaching the optimal solution in practice with arbitrarily small error is definitely more concrete. Problems that allow this stronger form of approximation are very important from the practical point of view and will be discussed in the Sect. 3.3.

Let us now consider MINIMUM PARTITION, that is, Problem 3.1: a simple approach to obtain an approximate solution for this problem is based on the greedy technique. Such an approach consists in sorting items in non-increasing order and then inserting them into set Y_1 or into set Y_2 according to the following rule: always insert the next item in the set of smaller overall weight (breaking ties arbitrarily). It is possible to show

that this procedure always returns a solution whose performance ratio is bounded by 6/5 and that this bound is indeed tight (see Exercise 3.9).

Program 3.3 basically consists in deriving an optimal solution of the subinstance including the k heaviest items and, subsequently, extending this solution by applying the greedy approach previously described. The next result shows that this algorithm provides a stronger approximation for MINIMUM PARTITION.

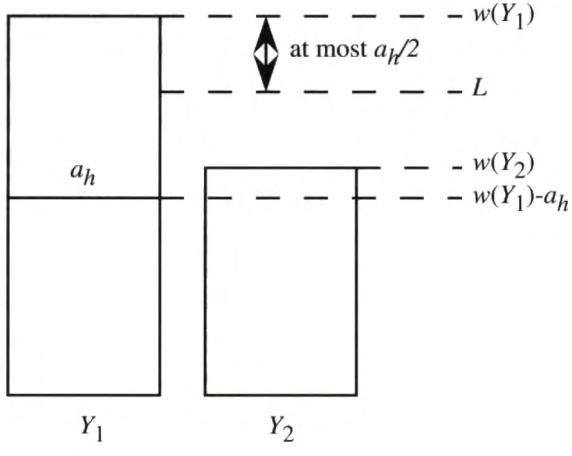


Figure 3.5
The analysis of Program 3.3

Theorem 3.8 ► *Program 3.3 is a polynomial-time approximation scheme for* MINIMUM PARTITION.

PROOF Let us first prove that, given an instance x of MINIMUM PARTITION and a rational $r > 1$, the algorithm provides an approximate solution (Y_1, Y_2) whose performance ratio is bounded by r. If $r \geq 2$, then the solution $(X, \emptyset)$ is clearly an r-approximate solution since any feasible solution has measure at least equal to half of the total weight $w(X) = \sum_{x_i \in X} a_i$. Let us then assume that $r < 2$ and let $w(Y_i) = \sum_{x_j \in Y_i} a_j$, for $i = 1, 2$, and $L = w(X)/2$. Without loss of generality, we may assume that $w(Y_1) \geq w(Y_2)$ and that x_h is the last item that has been added to Y_1 (see Fig. 3.5). This implies that $w(Y_1) - a_h \leq w(Y_2)$. By adding $w(Y_1)$ to both sides and dividing by 2 we obtain that

$$w(Y_1) - L \leq \frac{a_h}{2}.$$

If x_h has been inserted in Y_1 during the first phase of the algorithm, then it is easy to see that the obtained solution is indeed an optimal solution. Otherwise (that is, x_h has been inserted during the second phase), we have that $a_h \leq a_j$, for any j with $1 \leq j \leq k(r)$, and that $2L \geq a_h(k(r)+1)$. Since

Problem 3.2: Maximum Integer Knapsack

INSTANCE: Finite set X of types of items, for each $x_i \in X$, value $p_i \in Z^+$ and size $a_i \in Z^+$, positive integer b.

SOLUTION: An assignment $c : X \mapsto \mathbf{N}$ such that $\sum_{x_i \in X} a_i c(x_i) \leq b$.

MEASURE: Total value of the assignment, i.e., $\sum_{x_i \in X} p_i c(x_i)$.

$w(Y_1) \geq L \geq w(Y_2)$ and $m^*(x) \geq L$, the performance ratio of the computed solution is

$$\frac{w(Y_1)}{m^*(x)} \leq \frac{w(Y_1)}{L} \leq 1 + \frac{a_h}{2L} \leq 1 + \frac{1}{k(r)+1} \leq 1 + \frac{1}{\frac{2-r}{r-1}+1} = r.$$

Finally, we prove that the algorithm works in time $O(n \log n + n^{k(r)})$. In fact, we need time $O(n \log n)$ to sort the n items. Subsequently, the first phase of the algorithm requires time exponential in $k(r)$ in order to perform an exhaustive search for the optimal solution over the $k(r)$ heaviest items $x_1, \ldots, x_{k(r)}$ and all other steps have a smaller cost. Since $k(r)$ is $O(1/(r-1))$, the theorem follows. QED

3.2.1 The class PTAS

Let us now define the class of those NPO problems for which we can obtain an arbitrarily good approximate solution in polynomial time with respect to the size of the problem instance.

PTAS *is the class of* NPO *problems that admit a polynomial-time approximation scheme.*

◀ Definition 3.11
Class PTAS

The preceding result shows that MINIMUM PARTITION belongs to PTAS. Let us now see other examples of problems in PTAS. The first example will also show another application of the algorithmic technique of Program 3.3, which essentially consists of optimally solving a "subinstance" and, then, extending the obtained solution by applying a polynomial-time procedure.

Problem 3.2 models a variant of MAXIMUM KNAPSACK in which there is a set of types of items and we can take as many copies as we like of an item of a given type, provided the capacity constraint is not violated (observe that this problem is equivalent to Problem 2.8 with $d = 1$).

MAXIMUM INTEGER KNAPSACK *belongs to the class* PTAS.

◀ Theorem 3.9

Program 3.4: Integer Knapsack Scheme

input Set X of n types of items, values p_i, sizes a_i, $b \in \mathbb{N}$, rational $r > 1$;
output Assignment $c : X \mapsto \mathbb{N}$ such that $\sum_{x_i \in X} a_i c(x_i) \leq b$;
begin
  $\delta := \lceil \frac{1}{r-1} \rceil$;
  Sort types of items in non-increasing order with respect to their values;
  (* Let $(x_1, x_2, \ldots, x_n)$ be the sorted sequence *)
  for $i := 1$ **to** n **do** $c(x_i) := 0$;
  $F := \{f \mid f : X \mapsto \mathbb{N} \wedge \sum_{i=1}^{n} f(x_i) \leq \delta \wedge \sum_{i=1}^{n} a_i f(x_i) \leq b\}$;
  for all f in F **do**
  begin
    $k := $ maximum i such that $f(x_i) \neq 0$;
    $b_k := b - \sum_{i=1}^{k} a_i f(x_i)$;
    Let x_{md} be the type of items with maximal value/size ratio in $\{x_k, \ldots, x_n\}$;
    $f(x_{\mathrm{md}}) := f(x_{\mathrm{md}}) + \lfloor b_k / a_{\mathrm{md}} \rfloor$;
    if $\sum_{x_i \in X} p_i f(x_i) \geq \sum_{x_i \in X} p_i c(x_i)$ **then**
      for $i := 1$ **to** n **do** $c(x_i) := f(x_i)$;
  end;
  return c
end.

PROOF Given an instance I of MAXIMUM INTEGER KNAPSACK, we first notice that if we relax the integrality constraint on the values of function c (that is, if we allow fractions of items to be included in the knapsack), then the optimal solution can be easily computed as follows. Let x_{md} be a type of item with maximal value/size ratio: then, the optimal assignment for the relaxed problem is given by $c_{\mathrm{r}}(x_{\mathrm{md}}) = b/a_{\mathrm{md}}$ and $c_{\mathrm{r}}(x_i) = 0$ for all other types of items. For any type of item x, let $c(x) = \lfloor c_{\mathrm{r}}(x) \rfloor$: since $m^*(I) \leq b p_{\mathrm{md}} / a_{\mathrm{md}}$, we have that function c is a feasible solution of I such that the following holds:

$$m^*(I) - m(I,c) \leq b p_{\mathrm{md}} / a_{\mathrm{md}} - p_{\mathrm{md}} \lfloor b / a_{\mathrm{md}} \rfloor \leq p_{\mathrm{md}} \leq p_{\max}$$

where $p_{\max}$ is the maximal value.

The approximation scheme described in Program 3.4 makes use of the above observation in order to extend a partial solution. Let us first show that, for any instance I of MAXIMUM INTEGER KNAPSACK and for any $r > 1$, the algorithm indeed returns an r-approximate solution of I. Let c^* be an optimal solution of I. If $\sum_{x \in X} c^*(x) \leq \delta$, then $m(I,c) = m^*(I)$ where c is the solution returned by the algorithm.

Otherwise (that is, $\sum_{x \in X} c^*(x) > \delta$), let $(x_1, x_2, \ldots, x_n)$ be the sequence of types of items sorted in non-increasing order with respect to their values

and let c^*_δ be an assignment defined as follows:

$$c^*_\delta(x_i) = \begin{cases} c^*(x_i) & \text{if } \sum_{j=1}^{i} c^*(x_j) \leq \delta, \\ \delta - \sum_{j=1}^{i-1} c^*(x_j) & \text{if } \sum_{j=1}^{i-1} c^*(x_j) \leq \delta \text{ and } \sum_{j=1}^{i} c^*(x_j) > \delta, \\ 0 & \text{otherwise.} \end{cases}$$

Clearly, $c^*_\delta \in F$ since $\sum_{j=1}^{n} c^*_\delta(x_j) = \delta$. Let k be the maximum index i such that $c^*_\delta(x_i) \neq 0$, let $b_k = b - \sum_{i=1}^{k} a_i c^*_\delta(x_i)$, and let x_{md} be the type of items with maximal value/size ratio among $\{x_k, \ldots, x_n\}$. Then, $m(I,c) \geq m(I,c^*_\delta) + p_{\text{md}} \lfloor b_k/a_{\text{md}} \rfloor$ and $m^*(I) \leq m(I,c^*_\delta) + b_k p_{\text{md}}/a_{\text{md}}$. Therefore,

$$\begin{aligned} \frac{m^*(I)}{m(I,c)} &\leq \frac{m(I,c^*_\delta) + b_k p_{\text{md}}/a_{\text{md}}}{m(I,c^*_\delta) + p_{\text{md}} \lfloor b_k/a_{\text{md}} \rfloor} \\ &\leq \frac{m(I,c^*_\delta) + (b_k/a_{\text{md}} - \lfloor b_k/a_{\text{md}} \rfloor)\, p_{\text{md}}}{m(I,c^*_\delta)} \\ &= 1 + \frac{(b_k/a_{\text{md}} - \lfloor b_k/a_{\text{md}} \rfloor)\, p_{\text{md}}}{m(I,c^*_\delta)} \\ &\leq 1 + \frac{p_k}{\delta p_k} = \frac{\delta+1}{\delta}, \end{aligned}$$

where the last inequality is due to the fact that $(b_k/a_{\text{md}} - \lfloor b_k/a_{\text{md}} \rfloor)\, p_{\text{md}} \leq p_{\text{md}} \leq p_k$ and that $m(I,c^*_\delta) \geq \delta p_k$. From the definition of δ, it follows that the performance ratio is at most r.

Finally, let us estimate the running time of Program 3.4. Since $|F| = O(n^\delta)$ (see Exercise 3.10), the overall time is clearly $O(n^{1+\delta})$, that is, $O(n^{1+\frac{1}{r-1}})$. QED

The last example of a polynomial-time approximation scheme that we consider is a scheme for computing approximate solutions of MAXIMUM INDEPENDENT SET restricted to planar graphs. The algorithm is based on the fact that MAXIMUM INDEPENDENT SET is polynomial-time solvable when restricted to a special class of graphs, called k-outerplanar and defined below.

Given a planar embedding of a planar graph G, the *level* of a node is inductively defined as follows:

1. All nodes that lie on the border of the exterior face are at level 1.

2. For any $i > 1$, if we remove from the embedding all nodes of level j with $1 \leq j < i$, then all nodes (if any) that lie on the border of the exterior face of the remaining embedding are at level i.

A planar graph is said to be *k-outerplanar* if it admits a *k-outerplanar embedding*, that is, a planar embedding with maximal node level at most k.

Example 3.5 ► The embedding shown in Fig. 3.6 is a 6-outerplanar embedding. Indeed, for $i = 1, \ldots, 18$, the level of node v_i is $\lceil i/3 \rceil$.

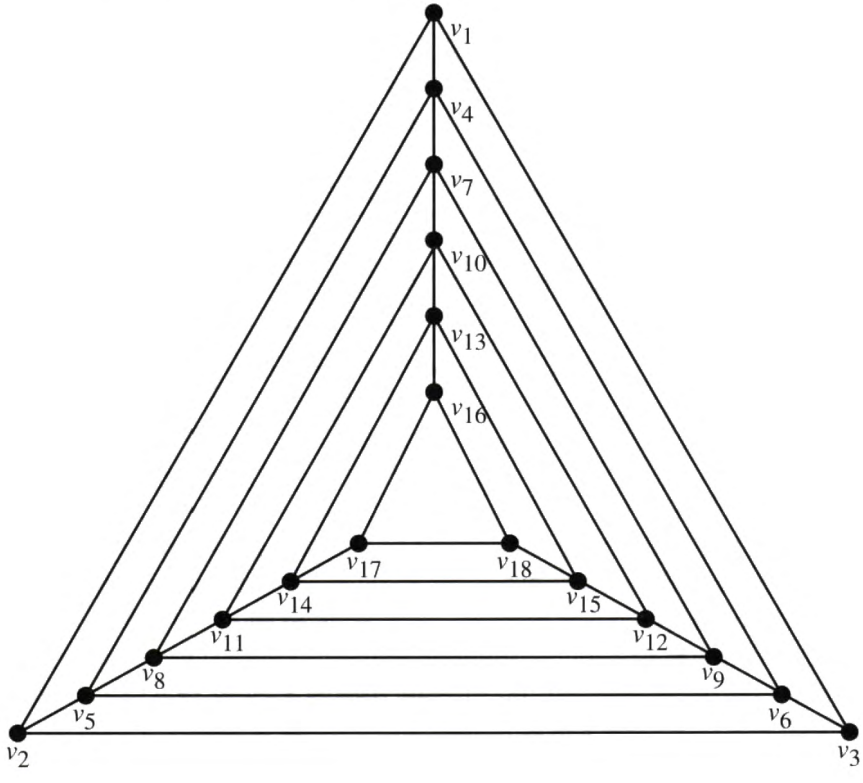


Figure 3.6
An example of a 6-outerplanar embedding

The following result, whose proof is here omitted (see Bibliographical notes), will be used by the approximation scheme.

Theorem 3.10 ► *For any k,* MAXIMUM INDEPENDENT SET *restricted to k-outerplanar graphs can be solved optimally in time* $O(8^k n)$ *where n is the number of nodes.*

The approximation scheme for MAXIMUM INDEPENDENT SET restricted to planar graphs exploits this result by considering a cover of the original graph formed by k-outerplanar graphs, where the parameter k depends on the required approximation ratio. In particular, the approximation scheme works in the following way (see Program 3.5).

Program 3.5: Independent Set Scheme

```
input Planar graph G = (V,E), rational r > 1;
output Independent set I ⊆ V;
begin
  k := ⌈1/(r-1)⌉;
  Compute a planar embedding of G (see Bibliographical notes);
  Compute node levels;
  (* Let V_i be the set of nodes of level i *)
  for i := 0 to k do
    begin
      Let V̄_i be the union of all sets V_j with j ≡ i (mod k+1);
      Let Ḡ_i be the subgraph of G induced by V − V̄_i;
      (* G_i is k-outerplanar *)
      Compute a maximal independent set I_i on Ḡ_i;
    end
  I := I_m such that |I_m| = max_{0≤i≤k}(|I_i|);
  return I
end.
```

Let r be the required approximation ratio and $k = \lceil 1/(r-1) \rceil$. Given a planar embedding of a planar graph G, for all i with $0 \leq i \leq k$, let $\overline{V}_i$ be the set of nodes whose level is congruent to i modulo $k+1$ and let $\overline{G}_i$ be the subgraph of G induced by all nodes not in $\overline{V}_i$. Since we have deleted at least one level every $k+1$ levels, it is easy to verify that G_i is a k-outerplanar graph: indeed, $\overline{G}_i$ is a collection of connected components, each with a k-outerplanar embedding. Therefore, we can compute the maximal independent set I_i of G_i in time $O(8^k n)$. Let I_m be the maximal cardinality independent set among $\{I_0, \ldots, I_k\}$.

If I^* denotes a maximal independent set of G, then there must exist an integer r with $0 \leq r \leq k$ such that $|\overline{V}_r \cap I^*| \leq |I^*|/(k+1)$. Hence, the maximal independent set I_r of $\overline{G}_r$ contains at least $k|I^*|/(k+1)$ nodes. Since $|I_m| \geq |I_r|$, we have that the performance ratio of I_m is at most $(k+1)/k \leq r$.

The running time of the algorithm is $O(8^k kn)$, since we apply $k+1$ times the exact algorithm for k-outerplanar graphs implied by Theorem 3.10. Hence, we have proved the following result.

MAXIMUM INDEPENDENT SET *restricted to planar graphs belongs to the class* PTAS. ◀ Theorem 3.11

Let us consider the graph G in Fig. 3.6 and assume we wish to find an independent set on G with performance ratio $3/2$. By applying the algorithm described in Program 3.5, we obtain $k = 2$, which implies that we obtain ◀ Example 3.6

three sets $\overline{V}_0 = \{v_7, v_8, v_9, v_{16}, v_{17}, v_{18}\}$, $\overline{V}_1 = \{v_1, v_2, v_3, v_{10}, v_{11}, v_{12}\}$, and $\overline{V}_2 = \{v_4, v_5, v_6, v_{13}, v_{14}, v_{15}\}$. Three possible corresponding maximal independent sets of $\overline{G}_i$ are $I_0 = \{v_1, v_5, v_{10}, v_{14}\}$, $I_1 = \{v_4, v_8, v_{13}, v_{18}\}$, and $I_2 = \{v_1, v_7, v_{11}, v_{16}\}$. Hence, we obtain that the solution returned by the algorithm contains 4 nodes. Clearly, $m^*(G) = 6$: indeed, we can choose exactly one node for every triangle formed by nodes of level i, for $i = 1, \ldots, 6$. As a consequence, we have that $m^*(G)/m(G,I) = 6/4 = r$.

3.2.2 APX versus PTAS

By definition, it is clear that the class PTAS is contained in the class APX. Henceforth, those problems that do not belong to APX, such as MINIMUM TRAVELING SALESPERSON, *a fortiori* cannot have a polynomial-time approximation scheme. A natural question hence arises at this point: Does there exist any NP optimization problem that for some value of r can be r-approximated in polynomial time, but which does not allow a polynomial-time approximation scheme? In other words the question is whether PTAS is *strictly* contained in APX. The answer is yes, provided that P $\neq$ NP. Again the main technique for proving the non-existence of a PTAS for an NPO problem is the gap technique.

Theorem 3.12 ▶ *If* P $\neq$ NP, *then* MINIMUM BIN PACKING *does not belong to the class* PTAS.

PROOF We show that, if P $\neq$ NP, then no r-approximate algorithm for MINIMUM BIN PACKING can be found with $r \leq 3/2 - \varepsilon$, for any $\varepsilon > 0$. To this aim, let us consider the PARTITION decision problem which consists in deciding whether a given set of weighted items can be partitioned into two sets of equal weight: this problem is NP-complete (see Bibliographical notes of Chap. 2). Given an instance x of PARTITION, let B be the sum of the weights of all items. We then define the corresponding instance x' of MINIMUM BIN PACKING as follows: for each item of x of weight w, x' has an item of size $2w/B$ (observe that we can always assume $w \leq B/2$ since, otherwise, x is a negative instance). If x is a positive instance, then $m^*(x') = 2$. Otherwise, $m^*(x') = 3$. From Theorem 3.7, it then follows the $3/2$ lower bound on the approximability of MINIMUM BIN PACKING. Therefore, unless P = NP, MINIMUM BIN PACKING does not admit a PTAS. QED

Since in Sect. 2.2.2 we have shown that MINIMUM BIN PACKING belongs to the class APX, the next corollary immediately follows.

Corollary 3.13 ▶ *If* P $\neq$ NP, *then* PTAS $\subset$ APX.

Other examples of problems that belong to APX but do not admit a PTAS are MAXIMUM SATISFIABILITY, MAXIMUM CUT, and MINIMUM METRIC TRAVELING SALESPERSON. Actually, the negative results on the existence of a PTAS for these problems make use of the gap technique in a more sophisticated way, as we will see in later chapters.

3.3 Fully polynomial-time approximation schemes

THE RUNNING time of the approximation schemes we have described in Chap. 2 and in this chapter depends on both the size of the input x and the inverse of the desired degree of approximation $r-1$: the better the approximation, the greater the running time. While, by definition of a PTAS, the running time must be polynomial in the size of the input x, the dependency on the quality of approximation may be very heavy. For example, in the case of MINIMUM PARTITION and MAXIMUM INDEPENDENT SET restricted to planar graphs, the dependency of the running time on the performance ratio has been shown to be exponential in $1/(r-1)$.

In some cases, such bad behavior strongly hampers the advantages of having a polynomial-time approximation scheme. In fact, the increase in the running time of the approximation scheme with the degree of approximation may prevent any practical use of the scheme.

3.3.1 The class FPTAS

A much better situation arises when the running time is polynomial *both* in the size of the input *and* in the inverse of the performance ratio.

◀ Definition 3.12
Fully polynomial-time approximation scheme

Let $\mathcal{P}$ be an NPO *problem. An algorithm $\mathcal{A}$ is said to be a* fully polynomial-time approximation scheme (FPTAS) *for $\mathcal{P}$ if, for any instance x of $\mathcal{P}$ and for any rational value $r > 1$, $\mathcal{A}$ with input (x,r) returns an r-approximate solution of x in time polynomial both in $|x|$ and in $1/(r-1)$.*

MAXIMUM KNAPSACK is an example of an optimization problem that admits an FPTAS: Program 2.9, in fact, is a fully polynomial-time approximation scheme for this problem, since its running time is $O(r|x|^3/(r-1))$.

◀ Definition 3.13
Class FPTAS

FPTAS *is the class of* NPO *problems that admit a fully polynomial-time approximation scheme.*

Clearly, the class FPTAS is contained in the class PTAS. Henceforth, those problems that do not belong to PTAS (such as MINIMUM TRAVELING SALESPERSON, which is not even in APX, or MINIMUM BIN PACKING, which is in APX but not in PTAS) *a fortiori* cannot have a fully polynomial-time approximation scheme.

The existence of a FPTAS for an NP-hard combinatorial optimization problems provides evidence that, for such a problem, despite the difficulty of finding an optimal solution, for most practical purposes the solution can be arbitrarily and efficiently approached.

3.3.2 The variable partitioning technique

In Sect. 2.5 we have shown how to derive a FPTAS for MAXIMUM KNAPSACK. Let us now consider the related MAXIMUM PRODUCT KNAPSACK problem in which the measure function is the product of the values of the chosen items. We will prove that also MAXIMUM PRODUCT KNAPSACK admits a FPTAS. In order to prove the result, we introduce a new technique, called *variable partitioning*, which is a variant of the fixed partitioning technique that we described in Sect. 2.5.

Observe that, in the case of MAXIMUM PRODUCT KNAPSACK, the value of the objective function can be as large as p_{max}^n, where p_{max} is the maximum value and n is the number of items. Therefore, if we make use of the fixed partitioning technique, it is possible to see that, when we divide the range into a polynomial number of equal size intervals (in order to obtain a polynomial-time algorithm), the relative error of the obtained solution cannot be bounded by any constant smaller than 1. The idea of the variable partitioning technique then consists in dividing the range of possible measures into a suitable collection of polynomially many *variable size* intervals.

More precisely, given an arbitrary instance x and given a rational $r > 1$, we divide the range of possible measures into the following intervals:

$$\{(0, \delta_1], (\delta_1, \delta_2], \ldots, (\delta_{t-1}, \delta_t]\},$$

where $\varepsilon = (r-1)/r$ and $\delta_j = (1 + \frac{\varepsilon}{2n})^j$ for $j = 1, \ldots, t$. Notice that, since the range of the possible measures is $(0, p_{max}^n]$, t is the smallest integer such that $(1+\varepsilon/(2n))^t \geq p_{max}^n$. Hence, t is $O(\frac{n^2}{\varepsilon} \log p_{max})$ and the algorithm has time complexity $O(\frac{n^3}{\varepsilon} \log p_{max})$.

Concerning the approximation, let Y be the solution returned by the algorithm and assume that $m(x, Y) \in (\delta_{i-1}, \delta_i]$: hence, the optimal measure must be contained in $(\delta_{i-1}, \delta_{i+n-1}]$. It is then possible to show that Y sat-

isfies the following inequality:

$$\frac{m^*(x) - m(x,Y)}{m^*(I)} \leq \varepsilon = \frac{r-1}{r}$$

(see Exercise 3.12), which implies that the performance ratio of Y is at most r. Thus the following theorem derives.

MAXIMUM PRODUCT KNAPSACK *belongs to the class* FPTAS. ◀ Theorem 3.14

3.3.3 Negative results for the class FPTAS

We now present some negative results which show that, unfortunately, many problems are not in FPTAS.

Actually, a first general result drastically reduces the class of combinatorial problems in PTAS which admit a FPTAS since it excludes the existence of a fully polynomial-time approximation scheme for all those optimization problems for which the value of the optimal measure is polynomially bounded with respect to the length of the instance. This result, in turn, will allow us to show that the containment of FPTAS in PTAS is proper.

An optimization problem is polynomially bounded *if there exists a polynomial p such that, for any instance x and for any $y \in \mathrm{SOL}(x)$, $m(x,y) \leq p(|x|)$.* ◀ Definition 3.14 *Polynomially bounded optimization problem*

No NP*-hard polynomially bounded optimization problem belongs to the class* FPTAS *unless* P = NP. ◀ Theorem 3.15

PROOF

Let $\mathcal{P}$ be an NP-hard polynomially bounded maximization problem (the minimization case would lead to a similar proof). Suppose we had a FPTAS $\mathcal{A}$ for $\mathcal{P}$ which, for any instance x and for any rational $r > 1$, runs in time bounded by $q(|x|, 1/(r-1))$ for a suitable polynomial q. Since $\mathcal{P}$ is polynomially bounded, there exists a polynomial p such that, for any instance x, $m^*(x) \leq p(|x|)$. If we choose $r = 1 + 1/p(|x|)$, then $\mathcal{A}(x,r)$ provides an optimal solution of x. Indeed, since $\mathcal{A}$ is a FPTAS, we have that

$$\frac{m^*(x)}{m(x, \mathcal{A}(x,r))} \leq r = \frac{p(|x|)+1}{p(|x|)},$$

that is,

$$m(x, \mathcal{A}(x,r)) \geq m^*(x)\frac{p(|x|)}{p(|x|)+1} = m^*(x) - \frac{m^*(x)}{p(|x|)+1} > m^*(x) - 1,$$

where the last inequality is due to the fact that $m^*(x) \leq p(|x|)$. From the integrality constraint on the measure function m, it follows that $m(x, \mathcal{A}(x,r)) = m^*(x)$, that is, $\mathcal{A}(x,r)$ is an optimal solution.

Since the running time of $\mathcal{A}(x,r)$ is bounded by $q(|x|, p(|x|))$, we have that $\mathcal{P}$ is solvable in polynomial time. From the NP-hardness of $\mathcal{P}$, the theorem thus follows. QED

Corollary 3.16 ▶ *If* P $\neq$ NP, *then* FPTAS $\subset$ PTAS.

PROOF As we have seen in Sect. 3.2.1, MAXIMUM INDEPENDENT SET restricted to planar graphs belongs to class PTAS. On the other side, this problem is clearly polynomially bounded and by the previous theorem it does not belong to the class FPTAS (unless P = NP). QED

3.3.4 Strong NP-completeness and pseudo-polynomiality

We conclude this section by giving another general condition that assures that a problem is not in FPTAS. Let us first introduce some definitions which are intrinsically interesting because they allow us to relate the approximability properties of a problem to its combinatorial structure. In particular, we are going to study the different ways in which numbers play a role in an NPO problem.

Let us consider, for example, MAXIMUM CUT. This problem is NP-hard: since no number appears in its instances (but the vertex indices), we may conclude that it is the combinatorial structure of MAXIMUM CUT, i.e. the property that the graph has to satisfy, that makes the problem hard.

When we consider other problems the situation is somewhat different. Let us consider MAXIMUM KNAPSACK. As we already noted in Sect. 2.5, by using a dynamic programming algorithm, we can solve this problem in time $O(n^2 p_{max})$: moreover, since p_{max} is an integer contained in the instance whose encoding requires $\lceil \log p_{max} \rceil$ bits, this algorithm is not a polynomial-time one. However, if we restrict ourselves to instances in which the numbers p_i have values bounded by a polynomial in the length of the instance, we obtain a polynomial-time algorithm. This means that the complexity of MAXIMUM KNAPSACK is essentially related to the size of the integers that appear in the input.

For any NPO problem $\mathcal{P}$ and for any instance x of $\mathcal{P}$, let $\max(x)$ denote the value of the largest number occurring in x. We note that, from a formal point of view, the definition of the function max depends on the encoding of the instance. However, we can repeat for the function max

the same kind of considerations that are usually made when considering the computational complexity of a problem assuming the length of the instance as the main parameter. In fact, if we choose two different functions max and max′ for the same problem, the results we are going to present do not change in the case that these two functions are polynomially related, that is, two polynomials p and q exist such that, for any instance x, both $\max(x) \leq p(\max'(x))$ and $\max'(x) \leq q(\max(x))$ hold.

For the NPO problems we are interested in, all the intuitive max functions we can think of are polynomially related. Thus, the concept of max is sufficiently flexible to be used in practice without any limitation.

◀ Definition 3.15
Pseudo-polynomial problem

An NPO *problem* $\mathcal{P}$ *is* pseudo-polynomial *if it can be solved by an algorithm that, on any instance* x*, runs in time bounded by a polynomial in* $|x|$ *and in* $\max(x)$.

◀ Example 3.7

In the case of MAXIMUM KNAPSACK, the max function can be defined as

$$\max(x) = \max(a_1, \ldots, a_n, p_1, \ldots, p_n, b).$$

The dynamic programming algorithm for MAXIMUM KNAPSACK thus shows that this problem is pseudo-polynomial. Indeed, for any instance x, the running time of this algorithm is $O(n^2 p_{max})$ and, hence, $O(n^2 \max(x))$.

The following result shows an interesting relationship between the concepts of pseudo-polynomiality and full approximability.

◀ Theorem 3.17

Let $\mathcal{P}$ *be an* NPO *problem in* FPTAS*. If a polynomial* p *exists such that, for every input* x*,* $m^*(x) \leq p(|x|, \max(x))$*, then* $\mathcal{P}$ *is a pseudo-polynomial problem.*

PROOF

Let $\mathcal{A}$ be a fully polynomial-time approximation scheme for $\mathcal{P}$. We will now exhibit an algorithm $\mathcal{A}'$ that solves any instance x of $\mathcal{P}$ in time polynomial both in $|x|$ and in $\max(x)$. This algorithm is simply defined as:

$$\mathcal{A}'(x) = \mathcal{A}\left(x, 1 + \frac{1}{p(|x|, \max(x)) + 1}\right).$$

Since the optimal measure is bounded by $p(|x|, \max(x))$, $\mathcal{A}'(x)$ must be an optimal solution. Regarding the running time of $\mathcal{A}'$, recall that $\mathcal{A}$ operates within time $q(|x|, 1/(r-1))$, for some polynomial q. Therefore, $\mathcal{A}'$ operates in time $q(|x|, p(|x|, \max(x)) + 1)$, that is, a polynomial in both $|x|$ and $\max(x)$.

QED

Let $\mathcal{P}$ be an NPO problem and let p be a polynomial. We denote by $\mathcal{P}^{\max,p}$ the problem obtained by restricting $\mathcal{P}$ to only those instances x for which $\max(x) \leq p(|x|)$. The following definition formally introduces the notion of an optimization problem whose computational hardness does not depend on the values of the numbers included in its instances.

Strongly NP*-hard problem*

Definition 3.16 ▶ *An* NPO *problem $\mathcal{P}$ is said to be* strongly NP-hard *if a polynomial p exists such that $\mathcal{P}^{\max,p}$ is* NP*-hard.*

Theorem 3.18 ▶ *If* P $\neq$ NP*, then no strongly* NP*-hard problem can be pseudo-polynomial.*

PROOF Let us assume that $\mathcal{P}$ is a strongly NP-hard problem, which is also pseudo-polynomial. This means that an algorithm exists that solves $\mathcal{P}$ in time $q(|x|, \max(x))$ for a suitable polynomial q. Then, for any polynomial p, $\mathcal{P}^{\max,p}$ can be solved in time $q(|x|, p(|x|))$. From the strong NP-hardness of $\mathcal{P}$, it also follows that a polynomial p exists such that $\mathcal{P}^{\max,p}$ is NP-hard. Hence, P = NP and the theorem follows. QED

Example 3.8 ▶ MAXIMUM CUT is an example of strongly NP-hard problem. Indeed, it is sufficient to consider the polynomial $p(n) = n$. Therefore, unless P = NP, MAXIMUM CUT is not pseudo-polynomial.

From Theorems 3.17 and 3.18, the following result can be immediately derived.

Corollary 3.19 ▶ *Let $\mathcal{P}$ be a strongly* NP*-hard problem that admits a polynomial p such that $m^*(x) \leq p(|x|, \max(x))$, for every input x. If* P $\neq$ NP*, then $\mathcal{P}$ does not belong to the class* FPTAS.

The concepts of pseudo-polynomiality and strong NP-hardness allow us to classify NPO problems in different classes. Once we have shown that an NPO problem is pseudo-polynomial, we can think that it is computationally easier than a problem that is strongly NP-hard (see Theorem 3.18). On the other hand, we can capture some connections of these concepts with the approximability properties. Also from this point of view, even if we only have partial relationships, it is clear that pseudo-polynomiality is linked to well-approximable problems (see Theorem 3.17) while strong NP-hardness seems to be one of the characteristics of problems that behave badly with respect to approximability (see Corollary 3.19).

3.4 Exercises

Exercise 3.1 Define a polynomial-time local search algorithm for MAXIMUM SATISFIABILITY. Prove that the algorithm finds a solution with measure at least one half of the optimum.

Program 3.6: Gavril

input Graph $G = (V, E)$;
output Vertex cover V';
begin
  repeat
    Choose any edge $e = (v_i, v_j) \in E$;
    $V' := V' \cup \{v_i, v_j\}$;
    $E := E - \{e' \mid e' \in E$ incident to v_i or $v_j\}$;
    until $E = \emptyset$;
  return V'
end.

Problem 3.3: Maximum k-Dimensional Matching

INSTANCE: Set $M \subseteq X_1 \times X_2 \times \ldots \times X_k$ where $X_1, X_2, \ldots, X_k$ are disjoint sets having the same number q of elements.

SOLUTION: Subset $M' \subseteq M$ such that no two elements of M' agree in any coordinate.

MEASURE: Cardinality of M'.

Exercise 3.2 Prove that Program 3.1 can be extended to the case in which each clause has an associated weight preserving the performance ratio.

Exercise 3.3 Show that Program 3.6, also known as *Gavril's algorithm*, is a 2-approximate algorithm for MINIMUM VERTEX COVER. (Hint: observe that the algorithm computes a maximal matching of the input graph.)

Exercise 3.4 Consider Problem 3.3. Show that, for any $k \geq 3$, MAXIMUM k-DIMENSIONAL MATCHING is k-approximable. (Hint: consider maximal matchings, that is, matchings that cannot be extended without violating the feasibility constraints.)

Exercise 3.5 Consider the following variant of Christofides' algorithm (known as the tree algorithm for MINIMUM TRAVELING SALESPERSON): after finding the minimum spanning tree T, create the multigraph M by using two copies of each edge of T. Show that this algorithm is 2-approximate and that the bound 2 is tight.

Exercise 3.6 Consider Problem 3.4. Prove that a minimum spanning tree on S is a 2-approximate solution for this problem.

Problem 3.4: Minimumum Metric Steiner Tree

INSTANCE: Complete graph $G=(V,E)$, edge weight function $w: E \mapsto \mathbb{N}$ satisfying the triangle inequality, and subset $S \subseteq V$ of required vertices.

SOLUTION: A Steiner tree T, i.e., a subgraph of G that is a tree and includes all the vertices in S.

MEASURE: The sum of the weights of the edges in T.

Problem 3.5: Minimum Knapsack

INSTANCE: Finite set X of items, for each $x_i \in X$, value $p_i \in Z^+$ and size $a_i \in Z^+$, positive integer b.

SOLUTION: A set of items $Y \subseteq X$ such that $\sum_{x_i \in Y} p_i \geq b$.

MEASURE: Total size of the chosen items, i.e., $\sum_{x_i \in Y} a_i$.

Exercise 3.7 Show that the greedy heuristic for MINIMUM VERTEX COVER based on repeatedly choosing a vertex with highest degree does not provide a constant approximation ratio.

Exercise 3.8 Prove that, for any NPO minimization problem $\mathcal{P}$, if there exists a constant k such that it is NP-hard to decide whether, given an instance x, $m^*(x) \leq k$, then no polynomial-time r-approximate algorithm for $\mathcal{P}$ with $r < (k+1)/k$ can exist, unless P = NP.

Exercise 3.9 Prove that the greedy algorithm for MINIMUM PARTITION described at the beginning of Sect. 3.2.1 is a polynomial-time 6/5-approximate algorithm. (Hint: use the proof of Theorem 3.8.)

Exercise 3.10 Prove that, for any integer c and for any rational $\delta > 0$, the number of ways of choosing c positive integers whose sum is less than δ is equal to $\binom{c+\lfloor\delta\rfloor}{\lfloor\delta\rfloor}$.

Exercise 3.11 By making use of a technique similar to the one used for MINIMUM PARTITION, show that, for any integer k, there is a $k/(k+1)$-approximate algorithm for MAXIMUM KNAPSACK.

Exercise 3.12 Fill in all the details of the proof of Theorem 3.14.

Exercise 3.13 Construct a FPTAS for MINIMUM KNAPSACK (see Problem 3.5) by making use of the variable partitioning technique.

Exercise 3.14 An NPO problem $\mathcal{P}$ is *simple* if, for every positive integer k, the problem of deciding whether an instance x of $\mathcal{P}$ has optimal measure at most k is in P. Prove that MAXIMUM CLIQUE is simple and that MINIMUM GRAPH COLORING is not simple (unless P = NP).

Exercise 3.15 Prove that a problem is simple if it belongs to the class PTAS.

Exercise 3.16 An NPO maximization problem $\mathcal{P}$ satisfies the *boundedness condition* if an algorithm $\mathcal{A}$ and a positive integer constant b exist such that the following hold: (a) for every instance x of $\mathcal{P}$ and for every positive integer c, $\mathcal{A}(x,c)$ is a feasible solution y of x such that $m^*(x) \leq m(x,y) + cb$, and (b) for every instance x of $\mathcal{P}$ and for every positive integer c, the time complexity of $\mathcal{A}(x,c)$ is a polynomial in $|x|$ whose degree depends only on the value $m(x,\mathcal{A}(x,c))/c$. Prove that MAXIMUM KNAPSACK verifies the boundedness condition.

Exercise 3.17 (*) Prove that an NPO maximization problem $\mathcal{P}$ admits a PTAS if and only if it is simple and satisfies the boundedness condition.

Exercise 3.18 An NPO problem $\mathcal{P}$ is *p-simple* if, for every positive integer k, the problem of deciding whether an instance x of $\mathcal{P}$ has optimal measure at most k is solvable in time bounded by a polynomial in $|x|$ and k. Prove that MAXIMUM KNAPSACK is p-simple.

Exercise 3.19 An NPO maximization problem $\mathcal{P}$ satisfies the *polynomial boundedness condition* if an algorithm $\mathcal{A}$ and a univariate polynomial p exist such that the following hold: (a) for every instance x of $\mathcal{P}$ and for every positive integer c, $\mathcal{A}(x,c)$ is a feasible solution y of x such that $m^*(x) \leq m(x,y) + cp(|x|)$, and (b) for every instance x of $\mathcal{P}$ and for every positive integer c, the time complexity of $\mathcal{A}(x,c)$ is a polynomial in $|x|$ whose degree depends only on the value $m(x,\mathcal{A}(x,c))/c$. Prove that MAXIMUM KNAPSACK verifies the polynomial boundedness condition.

Exercise 3.20 (*) Prove that an NPO maximization problem $\mathcal{P}$ admits a FPTAS if and only if it is p-simple and satisfies the polynomial boundedness condition.

3.5 Bibliographical notes

THE CONCEPT of approximation algorithm with "guaranteed performance" was introduced in the 1970s in the context of the first attempts to provide a formal analysis of computer heuristics for the solution

of difficult optimization problems. The problem of designing efficient algorithms capable of achieving "good" feasible solutions of optimization problems in those cases in which the exact solution could not be achieved unless running exponential-time algorithms, has, of course, been addressed since the beginning of computational studies in operations research. But it was not until the late 1960s that people started to perceive the need to go beyond computational experiments and to provide the formal analysis of the performance of an approximation algorithm in terms of quality of approximation. One of the first papers in which the performance af an approximation algorithm was analyzed is [Graham, 1966] which deals with multiprocessor scheduling.

In the subsequent years the study of approximation algorithms started to become more systematic. Among the papers that are considered to have laid down the first basic concepts relative to approximation algorithms, we may refer the reader to [Garey, Graham, and Ullman, 1972, Sahni, 1973, Johnson, 1974a, Nigmatullin, 1976]. In particular, in Johnson's paper the first examples of $f(n)$-approximate algorithms (that is with non-constant approximation ratio), and the first examples of approximation schemes are shown. In [Garey and Johnson, 1976b] the first survey with a complete view of the earliest results concerning approximation algorithms is provided.

In the late 1970s a large body of literature on approximation algorithms and on classes of approximability already existed. In [Horowitz and Sahni, 1978, Garey and Johnson, 1979] the authors provide the basic concepts and the related terminology (approximate algorithm, polynomial-time approximation scheme, fully polynomial-time approximation scheme). Those books have been the common ground for all work in the field.

The greedy algorithm for MAXIMUM SATISFIABILITY is due to [Johnson, 1974a]: a careful analysis of this algorithm is given in [Chen, Friesen, and Zheng, 1997] where it is shown that its performance ratio is at most 3/2. The approximation algorithm for MINIMUM VERTEX COVER based on the matching construction (see Exercise 3.3) is due to Gavril.

The proof that MINIMUM TRAVELING SALESPERSON is not approximable unless P = NP appears in [Sahni and Gonzalez, 1976] where the first examples of NP-hardness of approximation are presented. Note that such problems are called P-complete, a terminology never used afterwards with this meaning. The 2-approximate algorithm for the MINIMUM METRIC TRAVELING SALESPERSON based on the spanning tree construction (see Exercise 3.5) appears for the first time in [Korobkov and Krichevskii, 1966]. In [Rosenkrantz, Stearns, and Lewis, 1977] several heuristics for MINIMUM METRIC TRAVELING SALESPERSON are analyzed and various 2-approximate algorithms are shown. The 1.5-approximate algorithm

(that is, Christofides' algorithm) appears in [Christofides, 1976]. It is worth noting that in the metric case this is still the best result known in terms of approximation ratio.

The primal-dual algorithm for the minimum weight perfect matching which is used in the proof of Theorem 3.6is due to [Gabow, 1990]. Currently, the best known algorithm for finding a minimum weight maximal matching in a graph satisfying the triangular inequality is due to [Vaidya, 1990] and takes time $O(n^{2.5}(\log n)^4)$. Since in the case of Christofides' algorithm we have exactly this type of instances, we may also apply Vaidya's algorithm and the overall time of Christofides' algorithm becomes $O(n^{2.5}(\log n)^4)$.

The gap technique has been implicitly used for some time (for example, in the cited results on NP-hardness of approximation of MINIMUM TRAVELING SALESPERSON). The first proof of hardness of approximability (up to ratio 2) for the MINIMUM GRAPH COLORING problem, based on the gap technique, appeared in [Garey and Johnson, 1976a].

For a long time only a few problems admitting a PTAS were known. Among them there was MAXIMUM INDEPENDENT SET restricted to planar graphs. A PTAS for this problem was due to [Lipton and Tarjan, 1980] and, independently, to [Chiba, Nishizeki, and Saito, 1982]. In [Baker, 1994], by means of a new technique, it is proved a more general result for such problem. The author proved that the MAXIMUM INDEPENDENT SET can be solved in polynomial time for k-outerplanar graphs and, as a consequence, showed that several problems restricted to planar graphs admit a PTAS: beside MAXIMUM INDEPENDENT SET, such problems include MINIMUM VERTEX COVER and MINIMUM DOMINATING SET.

More recently, by means of new techniques, polynomial-time approximation schemes have been designed for a large group of geometric problems in the Euclidean plane by [Arora, 1997]. Among them, the most relevant are MINIMUM TRAVELING SALESPERSON and MINIMUM STEINER TREE. Notice that the result is remarkable, because in [Papadimitriou and Yannakakis, 1993] it is proved that in the general metric case MINIMUM TRAVELING SALESPERSON does not allow a PTAS.

One of the first examples of a fully polynomial-time approximation scheme, namely the FPTAS for the knapsack problem, was given by [Ibarra and Kim, 1975]. Both notions are extensively discussed in [Horowitz and Sahni, 1978] where more examples of problems in PTAS and FPTAS are shown.

The technique of variable partitioning used in the FPTAS for MAXIMUM PRODUCT KNAPSACK has been introduced in [Marchetti-Spaccamela and Romano, 1985].

The strict inclusion of FPTAS in the class PTAS was proved in [Korte, and

Schrader, 1981] by showing that MAXIMUM INTEGER m-DIMENSIONAL KNAPSACK (previously shown to be in PTAS by [Chandra, Hirschberg, and Wong, 1976]) is not in FPTAS. In the same paper, necessary and sufficient conditions for showing the existence of PTAS and FPTAS (based on generalizations of the dominance rules introduced for knapsack-type problems) were presented. Other necessary and sufficient conditions, of more general applicability, were provided by [Paz and Moran, 1981]. In [Ausiello, Marchetti-Spaccamela, and Protasi, 1980] it is shown that these conditions are strictly related to the notions of strong NP-completeness and pseudo-polynomiality introduced by [Garey and Johnson, 1978].

Finally it is worth noting that even though the standard approach to the study of approximation algorithms was based on the notions of relative error and performance ratio, as defined in Sect. 3.1.2, in [Hsu and Nemhauser, 1979, Ausiello, Marchetti-Spaccamela, and Protasi, 1980, Demange and Pascos, 1996] it is suggested to consider the so-called "differential performance measure" which relates the error made by an algorithm to the range of possible values of the measure.

Chapter 4

Input-Dependent and Asymptotic Approximation

In the previous two chapters we have seen examples of NPO problems that can be approximated either within a specific constant factor or within any constant factor. We also saw examples of NPO problems for which no approximation algorithm exists (unless P=NP) and examples of NPO problems for which an approximation algorithm but no approximation scheme exists (unless P=NP). To deal with these two latter kinds of problem, in this chapter we will relax the constraint on the performance ratio in two ways.

We will first allow the performance ratio to be dependent on the input size. Clearly, any NPO problem for which a solution can be computed in polynomial time is approximable with respect to this weaker notion of performance ratio. Indeed, since the measure function can be computed in polynomial time, the ratio between the optimal measure and the measure of *any* feasible solution is always bounded by a function exponential in the length of the input. However, our goal is to find algorithms that produce solutions whose ratio is bounded by a more slowly increasing function. In particular, we will provide an $O(\log n)$-approximate algorithm for MINIMUM SET COVER, where n denotes the cardinality of the set to be covered, an $O(n/\log n)$-approximate algorithm for MINIMUM GRAPH COLORING, where n denotes the number of nodes of the graph, and an $O(\log k)$-approximate algorithm for MINIMUM MULTI-CUT, where k denotes the number of pairs of vertices that have to be disconnected.

We will then consider *asymptotic approximation schemes*, that is, schemes whose performance ratio is bounded by a constant, for any in-

Problem 4.1: Minimum Set Cover

INSTANCE: Collection C of subsets of a finite set S.

SOLUTION: A set cover for S, i.e., a subset $C' \subseteq C$ such that every element in S belongs to at least one member of C'.

MEASURE: $|C'|$.

stance whose optimal measure is large enough. In this case, we will show that MINIMUM EDGE COLORING and MINIMUM BIN PACKING admit an asymptotic approximation scheme. We have already seen in Sect. 2.2.2 that the latter problem belongs to the class APX and in Sect. 3.2.2 that it does not belong to PTAS (unless P=NP). We will see in this chapter that the same holds for the former problem. In a certain sense, the existence of an asymptotic approximation scheme shows that these two problems are easier than other problems in APX – PTAS.

4.1 Between APX and NPO

AS WE already saw in the preceding chapters, for several problems (such as MINIMUM TRAVELING SALESPERSON) it is possible to prove that a constant performance ratio is not achievable unless P = NP. In these cases we can relax the constant requirement on the ratio by looking for algorithms whose performance depends on the length of the instance. In this section we will give three examples of these algorithms.

4.1.1 Approximating the set cover problem

The first problem we consider is MINIMUM SET COVER (see Problem 4.1). For this problem, let us consider Program 4.1, which is a simple polynomial-time greedy procedure to cover set S. At each iteration, the algorithm chooses the set that covers the maximum number of uncovered elements (breaking ties arbitrarily) and updates the remaining sets.

Theorem 4.1 ▶ *Program 4.1 is a* $(\lfloor \ln n \rfloor + 1)$*-approximate algorithm for* MINIMUM SET COVER, *where* n *denotes the number of elements of the universe* S.

PROOF Given an instance of MINIMUM SET COVER, let k denote the largest cardinality of the sets in the collection. We will now prove that, for any optimal

Program 4.1: Greedy Set Cover

input Collection C of subsets of a finite set S;
output Set cover C';
begin
  $U := S$;
  for each set c_i in C **do** $c_i' := c_i$;
  $C' := \emptyset$;
  while $U \neq \emptyset$ **do**
  begin
    Let c_j' be a set with maximum cardinality;
    $C' := C' \cup \{c_j\}$;
    $U := U - c_j'$;
    for each c_i' **do** $c_i' := c_i' - c_j'$
  end;
  return C'
end.

solution C^*,

$$\sum_{c_i \in C^*} \mathcal{H}(|c_i|) \geq |C'|, \tag{4.1}$$

where, for any integer $r > 0$, $\mathcal{H}(r)$ denotes the r-th harmonic number (that is, $\mathcal{H}(r) = \sum_{i=1}^{r} 1/i$) and C' is the set cover obtained by Program 4.1. Since $|c_i| \leq k$, for any i with $1 \leq i \leq m$, $\mathcal{H}(k) \leq \lfloor \ln k \rfloor + 1$, and $k \leq n$, the above inequality implies that

$$|C'| \leq \sum_{c_i \in C^*} \mathcal{H}(k) \leq \mathcal{H}(n)|C^*| \leq (\lfloor \ln n \rfloor + 1)\,|C^*|.$$

That is, the performance ratio of Program 4.1 is at most $\lfloor \ln n \rfloor + 1$.

For any instance $x = (S, \{c_1, \ldots, c_m\})$ of MINIMUM SET COVER, let us denote by $a_1, \ldots, a_{|C'|}$ the sequence of indices of the subsets that belong to the set cover C'. Moreover, for any $j \in \{1, \ldots, |C'|\}$ and for any $i \in \{1, \ldots, m\}$, let c_i^j be the *surviving* part of c_i before index a_j has been chosen (that is, the subset of c_i that does not intersect any chosen set). Clearly, $c_i^1 = c_i$, for any i. Moreover, for the sake of simplicity, we will adopt the convention that $c_i^{|C'|+1} = \emptyset$, for any $i \in \{1, \ldots, m\}$. The set of elements of c_i that are covered for the first time by c_{a_j} is given by

$$c_i \cap c_{a_j}^j = c_i^j \cap c_{a_j}^j = c_i^j - c_i^{j+1}. \tag{4.2}$$

For any $i \in \{1, \ldots, m\}$, let l_i denote the largest index such that $|c_i^{l_i}| > 0$: that is, after $c_{a_{l_i}}$ has been included into C' the subset c_i has been covered.

The proof of Eq. (4.1) consists of the following two steps:

1. Prove that, for any $i \in \{1,\ldots,m\}$,
$$\mathcal{H}(|c_i|) \geq \sum_{j=1}^{|C'|} \frac{|c_i \cap c_{a_j}^j|}{|c_{a_j}^j|}.$$

2. Prove that, for any set cover C'' (and, therefore, for any optimal solution),
$$\sum_{c_i \in C''} \sum_{j=1}^{|C'|} \frac{|c_i \cap c_{a_j}^j|}{|c_{a_j}^j|} \geq |C'|.$$

Both steps consist of simple algebraic calculations. For what regards the first step, notice that, for any i with $1 \leq i \leq m$,

$$\begin{aligned}
\sum_{j=1}^{|C'|} \frac{|c_i \cap c_{a_j}^j|}{|c_{a_j}^j|} &= \sum_{j=1}^{|C'|} \frac{|c_i^j| - |c_i^{j+1}|}{|c_{a_j}^j|} \leq \sum_{j=1}^{l_i} \frac{|c_i^j| - |c_i^{j+1}|}{|c_i^j|} \\
&= \sum_{j=1}^{l_i} \sum_{k=|c_i^{j+1}|+1}^{|c_i^j|} \frac{1}{|c_i^j|} \leq \sum_{j=1}^{l_i} \sum_{k=1}^{|c_i^j|-|c_i^{j+1}|} \frac{1}{k+|c_i^{j+1}|} \\
&\leq \sum_{j=1}^{l_i} \left(\mathcal{H}(|c_i^j|) - \mathcal{H}(|c_i^{j+1}|)\right) = \mathcal{H}(|c_i^1|) = \mathcal{H}(|c_i|)
\end{aligned}$$

where the first equality follows from Eq. (4.2) and the first inequality is due to the fact that, for any j with $1 \leq j \leq |C'|$, $|c_i^j| \leq |c_{a_j}^j|$ (since the algorithm always chooses the biggest surviving set).

The second step follows as easily as the first one. Namely, for any set cover C'',

$$\sum_{c_i \in C''} \sum_{j=1}^{|C'|} \frac{|c_i \cap c_{a_j}^j|}{|c_{a_j}^j|} = \sum_{j=1}^{|C'|} \frac{1}{|c_{a_j}^j|} \sum_{c_i \in C''} |c_i \cap c_{a_j}^j| \geq \sum_{j=1}^{|C'|} \frac{|c_{a_j}^j|}{|c_{a_j}^j|} = |C'|$$

where the inequality is due to the fact that C'' is a set cover. In conclusion, Eq. (4.1) is satisfied and the theorem follows.

QED

Example 4.1 ▶ Let us consider the instance of MINIMUM SET COVER with $S = \{1,\ldots,24\}$ and with C given by the following seven subsets of S: $c_1 = \{1,\ldots,8\}$, $c_2 = \{9,\ldots,16\}$, $c_3 = \{17,\ldots,24\}$, $c_4 = \{5,6,7,8,13,14,15,16,21,22,23,24\}$, $c_5 = \{3,4,11,12,19,20\}$, $c_6 = \{2,10,18\}$, $c_7 = \{1,9,17\}$. The first subset chosen by Program 4.1 is c_4. As a consequence, the first three subsets become $c_1^2 = \{1,\ldots,4\}$, $c_2^2 = \{9,\ldots,12\}$, and $c_3^2 = \{17,\ldots,20\}$, respectively. The next subset chosen is c_5 and, once again, the algorithm modifies the first three subsets that become $c_1^3 = \{1,2\}$, $c_2^3 = \{9,10\}$, and $c_3^3 = \{17,18\}$, respectively. At the next two iterations of the loop, the subsets chosen are c_6 and c_7, respectively, so that the solution computed by the algorithm uses four subsets. On the other hand, it is easy to verify that the first three subsets form an optimal solution.

Program 4.2: Greedy Graph Coloring

```
input Graph G = (V,E);
output Node coloring of G;
begin
  i := 0;
  U := V;
  while U ≠ ∅ do
  begin
    i := i + 1;
    W := U;
    while W ≠ ∅ do
    begin
      H := graph induced by W;
      v := node of minimum degree in H;
      f(v) := i;
      W := W − {v} − {z|z is a neighbor of v in H}
    end;
    U := U − f^{-1}(i);
  end;
  return f
end.
```

The analysis of Theorem 4.1 is tight: indeed, it is possible to generalize the previous example in order to prove such statement (see Exercise 4.1).

4.1.2 Approximating the graph coloring problem

In this section we provide an $O(n/\log n)$-approximate algorithm for MINIMUM GRAPH COLORING based on repeatedly finding an independent set (i.e., a set of vertices that can be colored with one color) and coloring it with a new color, until all vertices have been colored. This algorithm is given as Program 4.2 and, clearly, runs in polynomial time. The next result gives an upper bound on its performance ratio.

◀ Lemma 4.2

Program 4.2 colors any k-colorable graph $G = (V,E)$ with at most $3|V|/\log_k |V|$ colors.

PROOF

Let H be the graph considered by the algorithm at the beginning of a generic iteration of the inner loop and let W be the corresponding set of vertices. Since H is a subgraph of a k-colorable graph, H is a k-colorable graph and, therefore, it must contain an independent set of size at least $|W|/k$. Each vertex of this set has degree at most $|W| - |W|/k =$

$|W|(k-1)/k$. This implies that the minimum degree of H is at most $|W|(k-1)/k$ so that at least $|W| - |W|(k-1)/k = |W|/k$ nodes will still be in W at the beginning of the next iteration. Since the inner loop ends when W becomes empty, we have that at least $\lceil \log_k |W| \rceil$ iterations have to be performed, which in turn implies that, at the end of the inner loop,

$$|\{v \mid v \in W \wedge f(v) = i\}| \geq \lceil \log_k |W| \rceil.$$

Hence, for each used color i, the number of vertices colored with i is at least $\lceil \log_k |U| \rceil$, where U denotes the set of nodes uncolored just before color i is considered.

Let us now analyze the size of U at the beginning of an iteration of the outer loop. If $|U| \geq |V|/\log_k |V|$, then we have that

$$\lceil \log_k |U| \rceil \geq \log_k |U| \geq \log_k (|V|/\log_k |V|) > \log_k \sqrt{|V|} = \frac{1}{2} \log_k |V|.$$

This implies that the size of U decreases by at least $\frac{1}{2} \log_k |V|$ at each iteration: as a consequence, the first time $|U|$ becomes smaller than $|V|/\log_k |V|$, the algorithm has used no more than $2|V|/\log_k |V|$ colors.

If $|U| < |V|/\log_k |V|$, it is clear that the algorithm colors all remaining nodes in U with at most $|U| < |V|/\log_k |V|$ colors. It follows that the algorithm uses at most $3|V|/\log_k |V|$ colors.

QED

Theorem 4.3 ▶ *MINIMUM GRAPH COLORING admits an $O(n/\log n)$-approximate algorithm, where n denotes the number of nodes.*

PROOF The previous lemma implies that, for any input graph G with n nodes, the solution returned by Program 4.2 uses at most $3n/\log_{m^*(G)} n = 3n \log(m^*(G))/\log n$ colors, where $m^*(G)$ denotes the minimum number of colors necessary to color G. This implies that the performance ratio of this algorithm is at most

$$\frac{3n \log(m^*(G))/\log n}{m^*(G)} = O(n/\log n)$$

QED and the theorem follows.

Example 4.2 ▶ Let us consider the graph of Fig. 4.1(a). In this case $m^*(G) = 3$: indeed, we can assign to nodes a, d, and h the first color, to nodes b, e, and g the second color, and to nodes c and f the third color. Program 4.2, instead, could first choose node a and assign to it the first color. The resulting graph H is shown in Fig. 4.1(b): at this step, node e is chosen and the first color is assigned to it. The new graph H is shown in Fig. 4.1(c): in this case, the algorithm could choose node h and assign to it the first color. In conclusion, after exiting the inner loop, nodes a, e, and h have been assigned the first color.

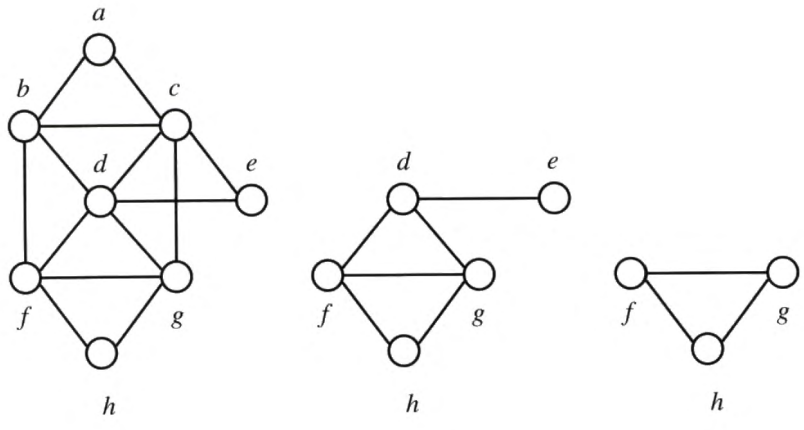


Figure 4.1
The first iteration of the outer loop of Program 4.2

Since uncolored nodes exist (see Fig. 4.2(a)), the algorithm will continue executing the outer loop and a possible run of the algorithm could assign the second color to nodes b and g. Successively, the third color could be assigned to nodes c and f and, finally, the fourth color is assigned to node d (see Figs. 4.2(a) and (b)). Hence, in this case the algorithm produces a 4-coloring.

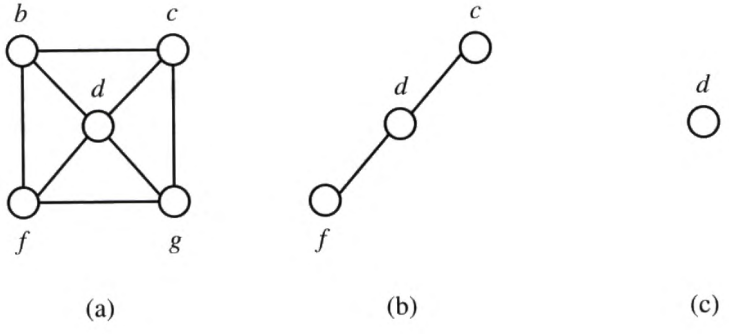


Figure 4.2
The next steps of Program 4.2

4.1.3 Approximating the minimum multi-cut problem

In this section, we give another example of how duality relationships between integer linear programming problems (see Appendix A) can be exploited to derive efficient approximation algorithms (a first example of an approximation algorithm based on the duality relationship has been given in Sect. 2.4.2).

First of all, let us recall that MAXIMUM FLOW (see Problem 4.2) consists in finding the maximum flow that can be routed in a network from a

Problem 4.2: Maximum Flow

INSTANCE: Graph $G = (V,E)$, edge capacity function $c : E \mapsto \mathbb{Q}^+$, pair $(s,t) \in V \times V$.

SOLUTION: A flow from s to t, that is, a function $f : V \times V \mapsto \mathbb{Q}^+$ such that:

1. $f(u,v)$ is defined if and only if $(u,v) \in E$;
2. $f(u,v) + f(v,u) \leq c(u,v)$, for each $(u,v) \in E$, that is, the total flow in an edge cannot exceed its capacity;
3. $\sum_{u \in V-\{v\}} f(u,v) = \sum_{u \in V-\{v\}} f(v,u)$ for each $v \in V - \{s,t\}$, that is, the flow entering in a node must equate the flow exiting the same node, for all nodes except the source and the destination.

MEASURE: $\sum_{u \in V-\{t\}} f(u,t) - \sum_{u \in V-\{t\}} f(t,u)$, that is, the total flow entering the destination.

source s to a destination t without violating some specified edge capacity constraints. Moreover, MINIMUM CUT (see Problem 4.3) requires to find the set of arcs of minimum total capacity whose removal disconnects the source from the destination in a network. These two problems are related through the maxflow-mincut theorem (see Bibliographical notes), a central result in flow theory, that lies at the basis of the solution of many relevant optimization problems. Such theorem states that, since a duality relationship holds between MAXIMUM FLOW and MINIMUM CUT, their optimal solutions have the same value.

Let us now consider the generalization of MAXIMUM FLOW to the case in which more than one commodity has to flow in the network (see Problem 4.4). This problem has the following linear programming formulation LP_{MMF}:

$$\begin{aligned}
\texttt{maximize} \quad & \sum_{i=1}^{k} \sum_{v \in V-\{t_i\}} (f^i_{v,t_i} - f^i_{t_i,v}) & \\
\texttt{subject to} \quad & \sum_{(v,u) \in E} f^i_{vu} - \sum_{(u,v) \in E} f^i_{uv} = 0 & \forall v \in V - \{s_i,t_i\},\ i = 1,\ldots,k \\
& \sum_{i=1}^{k} f^i_{uv} + \sum_{i=1}^{k} f^i_{vu} \leq c_{uv} & \forall (u,v) \in E \\
& f^i_{uv} \geq 0 & \forall (u,v) \in E,
\end{aligned}$$

where f^i_{uv} is the flow of commodity i along edge (u,v) from u to v) and c_{uv} denotes the capacity of edge (u,v). The first set of constraints represents

Problem 4.3: Minimum Cut

INSTANCE: Graph $G = (V,E)$, edge capacity function $c : E \mapsto \mathbb{Q}^+$, pair $(s,t) \in V \times V$.

SOLUTION: A set $E' \subseteq E$ of edges whose removal disconnects s and t.

MEASURE: The overall capacity of E', i.e., $\sum_{e \in E'} c(e)$.

flow conservation at vertices, while the second set of inequalities states that the total flow value assigned to each edge does not exceed its capacity.

Let us also consider the corresponding generalization of MINIMUM CUT, shown in Problem 4.5. This problem has the following integer linear programming formulation:

$$\begin{array}{lll} \texttt{minimize} & \displaystyle\sum_{(u,v)\in E} d_{uv}c_{uv} & \\ \texttt{subject to} & \displaystyle\sum_{(u,v)\in E} d_{uv}\pi_i^j(u,v) \geq 1 & \forall \pi_i^j \in \mathcal{P}_i, i = 1,\ldots,k \\ & d_{uv} \in \mathbf{N} & \forall (u,v) \in E, \end{array}$$

where $\mathcal{P}_i$ denotes the set of all paths connecting s_i to t_i, π_i^j is the j-th of such paths, $\pi_i^j(u,v)$ is 1 if (u,v) is an edge in π_i^j, and $\pi_i^j(u,v)$ is 0 otherwise. Notice that, indeed, any solution of the above problem corresponds to a multi-cut, defined as the set of edges (u,v) such that $d_{uv} > 0$. Notice also that the above formulation has an exponential number of constraints: however, an equivalent, less natural, formulation ILP_{MC} with polynomially many constraints is the following one (see Exercise 4.8):

$$\begin{array}{lll} \texttt{minimize} & \displaystyle\sum_{(u,v)\in E} d_{uv}c_{uv} & \\ \texttt{subject to} & d_{uv} \geq p_u^i - p_v^i & \forall (u,v) \in E,\ i = 1,\ldots,k \\ & p_s^i - p_t^i \geq 1 & i = 1,\ldots,k \\ & p_v^i \in \mathbf{N} & \forall v \in V,\ i = 1,\ldots,k \\ & d_{uv} \in \mathbf{N} & \forall (u,v) \in E, \end{array}$$

where d_{uv} is a variable that, by the minimization requirement, will assume value at most 1 in the optimal solution and with such a value will denote that edge (u,v) is in the multi-cut. Intuitively, variables p_v^i denote a *potential* associated to flow i at vertex v. In this way, the constraints impose that, for each pair (s_i,t_i), they must be sufficiently "far apart" to always have at least one edge in the multi-cut between them.

It is possible to see that, by applying the well-known primal-dual transformation (see Appendix A) to LP_{MMF}, we obtain the same linear program

Problem 4.4: Maximum Multi-commodity Flow

INSTANCE: Graph $G = (V,E)$, edge capacity function $c : E \mapsto \mathbb{Q}^+$, k pairs $(s_i,t_i) \in V \times V$.

SOLUTION: A set of k flows f^i, each from s_i to t_i, for $i = 1,\ldots,k$, such that the total flow value $\sum_{\forall i} f^i(u,v)$ that passes through edge (u,v) does not exceed its capacity.

MEASURE: The sum of the total flows entering all destinations.

Problem 4.5: Minimum Multi-Cut

INSTANCE: Graph $G = (V,E)$, edge capacity function $c : E \mapsto \mathbb{Q}^+$, k pairs $(s_i,t_i) \in V \times V$.

SOLUTION: A subset $E' \subseteq E$ of edges whose removal disconnects s_i and t_i, $i = 1,\ldots,k$.

MEASURE: The overall capacity $\sum_{e \in E'} c(e)$ of E'.

as by relaxing the integrality constraints of ILP_{MC}. Such a program is the linear programming formulation of MINIMUM FRACTIONAL MULTI-CUT. This situation is similar to what happens in the case of MAXIMUM FLOW, whose dual is MINIMUM FRACTIONAL CUT, i.e., the relaxation of MINIMUM CUT to include also non integral solutions. However, the combinatorial structure of MINIMUM FRACTIONAL CUT guarantees that any optimal solution is an integral one and, as a consequence, an optimal solution also for MINIMUM CUT. Unfortunately, the equality does not hold for MINIMUM FRACTIONAL MULTI-CUT and all we can say in this case is that the capacity of the minimum multi-cut in a graph is *at least* the value of the maximum multi-commodity flow in the same graph.

Actually, a stronger relationship has been shown between MINIMUM MULTI-CUT and MAXIMUM MULTI-COMMODITY FLOW: given a graph $G = (V,E)$ with k commodities, the capacity of the minimum multi-cut is at most $O(\log k)$ times the maximum flow in G.

In the following, we present an approximation algorithm that, given an instance of MINIMUM MULTI-CUT, returns a solution whose capacity is bounded by $O(\log k)$ times the maximum multi-commodity flow in the graph. This guarantees that the value of such a solution is at most $O(\log k)$ times the capacity of the minimum multi-cut. The algorithm exploits the knowledge of an optimal fractional multi-cut, which can be efficiently computed by solving in polynomial time either MINIMUM FRACTIONAL

MULTI-CUT or its dual MAXIMUM MULTI-COMMODITY FLOW.

Let $d : E \mapsto \mathbb{Q}^+$ be a function that associates to each edge (u,v) the value of variable d_{uv} in the optimal solution of MINIMUM FRACTIONAL MULTI-CUT. If we consider the graph $G = (V,E)$ with functions $c : E \mapsto \mathbb{Q}^+$ and $d : E \mapsto \mathbb{Q}^+$ and k pairs $\{(s_1,t_1),\ldots,(s_k,t_k)\}$, we may see it as a set of pipes (edges) connected at nodes, where pipe (u,v) has length $d(u,v)$ and cross section $c(u,v)$ (and thus volume $c(u,v)d(u,v)$). In particular, this pipe system is the one of minimum volume such that, for each pair (s_i,t_i), the distance between s_i and t_i is at least 1. Moreover, let $\Psi = \sum_{(u,v)\in E} c(u,v)d(u,v)$ be the overall volume of G: clearly, since MINIMUM FRACTIONAL MULTI-CUT is a relaxation of MINIMUM MULTI-CUT, Ψ is a lower bound on the optimal measure of the corresponding MINIMUM MULTI-CUT problem.

The algorithm (see Program 4.3) works by iteratively producing a sequence $\mathcal{V} = (V_1,V_2,\ldots,V_q)$ of disjoint subsets of V such that, for each $r = 1,\ldots,q$:

1. at most one between s_i and t_i is contained in V_r, for each $i = 1,\ldots,k$;
2. there exists $l \in \{1,\ldots,k\}$ such that either s_l or t_l is contained in V_r.

Note that, for any r with $1 \leq r \leq q$, the subsequence $(V_1,\ldots,V_r)$ *separates* at least r pairs (s_i,t_i).

The algorithm works in phases, where at each phase a new set V_r is added to $\mathcal{V}$, until each vertex is included in one subset of $\mathcal{V}$. In particular, let $\overline{G}_r = (\overline{V}_r,\overline{E}_r)$ be the graph induced by $\overline{V}_r = V - (\cup_{j=1}^{r-1} V_j)$: if there exists no pair (s_i,t_i) such that $s_i \in \overline{V}_r$ and $t_i \in \overline{V}_r$, then the algorithm sets $V_r = \overline{V}_r$ and returns the multi-cut $\{(u,v) \in E \mid u \in V_p \wedge v \in V_q \wedge p \neq q\}$.

Otherwise, let (s_i,t_i) be an arbitrarily chosen pair such that both $s_i \in \overline{V}_r$ and $t_i \in \overline{V}_r$. The algorithm then computes a suitable set $V_r \subseteq \overline{V}_r$ such that exactly one between s_i and t_i is contained in V_r. This is performed by selecting a set of vertices that are within a certain distance ρ from s_i. This set of vertices is a *ball* centered at s_i and of radius ρ. The algorithm starts with $\rho = 0$ and iteratively increases ρ until a suitable condition is verified.

In order to describe more in detail how this is done, let us first introduce some definitions. For any $u,v \in V$, let $\delta(u,v)$ be the length of a shortest path from u to v, with respect to edge length $d : E \mapsto \mathbb{Q}^+$. For any $v \in \overline{V}_r$ and for any $\rho \in \mathbb{Q}$, the *ball* of radius ρ around v (with respect to δ) is defined as

$$\mathcal{B}_\delta(v,\rho) = \{u \in \overline{V}_r \mid \delta(u,v) \leq \rho\}.$$

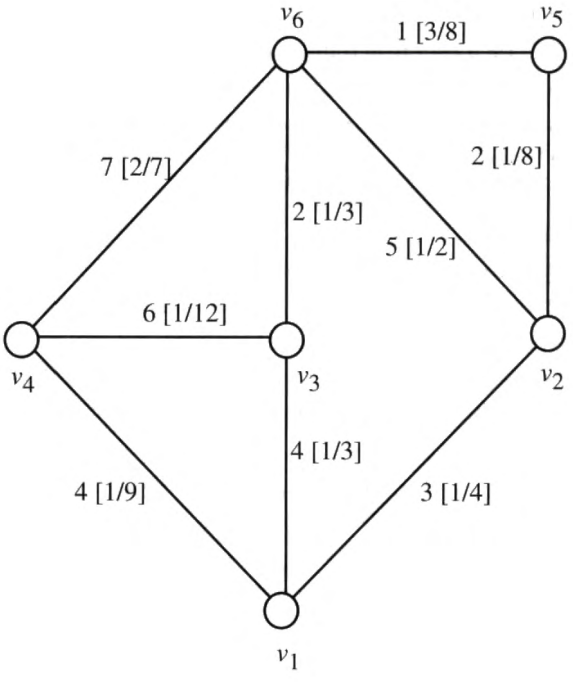


Figure 4.3
Sample graph G: square brackets contain the values of function d

Moreover, let

$$\mathcal{E}_\delta(v,\rho) = \{(w,z) \in \overline{E}_r \mid w \in B_\delta(v,\rho) \wedge z \in B_\delta(v,\rho)\}$$

be the set of edges completely contained in $\mathcal{B}_\delta(v,\rho)$ and let

$$\overline{\mathcal{E}}_\delta(v,\rho) = \{(w,z) \in \overline{E}_r - \mathcal{E}_\delta(v,\rho) \mid w \in B_\delta(v,\rho) \vee z \in B_\delta(v,\rho)\}$$

be the set of edges partially contained in $\mathcal{B}_\delta(v,\rho)$. Finally, the *volume* of $\mathcal{B}_\delta(v,\rho)$ is defined as

$$V_\delta(v,\rho) = \frac{\Psi}{k} + \sum_{(w,z)\in\mathcal{E}_\delta(v,\rho)} c(w,z)d(w,z) + \sum_{(w,z)\in\overline{\mathcal{E}}_\delta(v,\rho)} c(w,z)(\rho - \min(\delta(v,w),\delta(v,z))).$$

while the *cost* of $\mathcal{B}_\delta(v,\rho)$ is defined as

$$C_\delta(v,\rho) = \sum_{(w,z)\in\overline{E}_\delta(v,\rho)} c(w,z).$$

Note that, apart from a fixed amount of volume Ψ/k (that we assume located on v), each edge contributes to the volume of $\mathcal{B}_\delta(v,\rho)$ for its fraction contained in $\mathcal{B}_\delta(v,\rho)$. On the other side, an edge contributes to the cost of $\mathcal{B}_\delta(v,\rho)$ for its capacity, if it is in the cut separating $\mathcal{B}_\delta(v,\rho)$ from $V - \mathcal{B}_\delta(v,\rho)$.

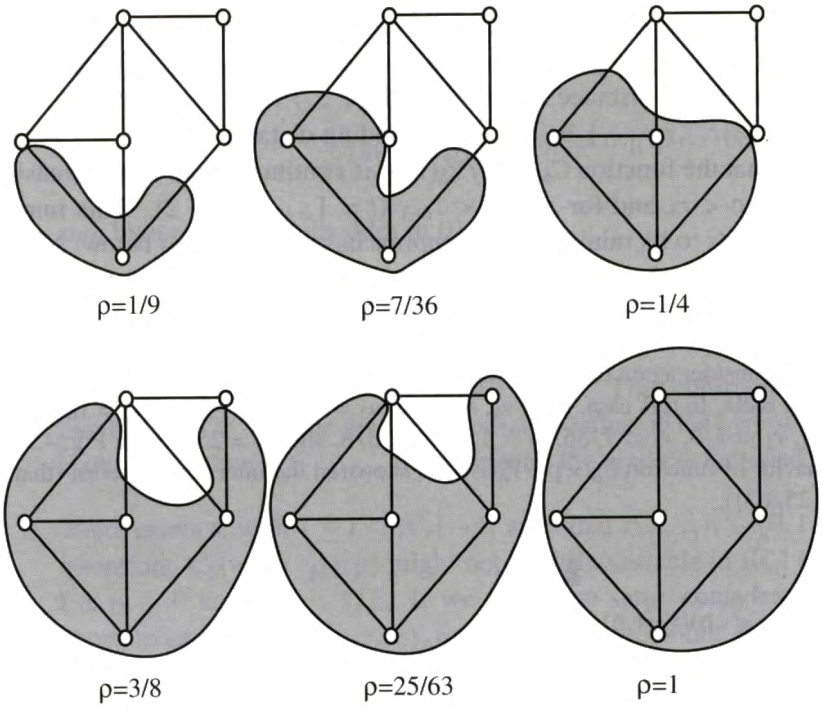


Figure 4.4
The balls $B_\delta(v_1,\rho)$ for some values of ρ

◀ Example 4.3

Let us consider the graph shown in Fig. 4.3, where the square brackets contain the values of function d, and assume that $k=2$. Hence, the amount of volume that the algorithm assumes to be located at each node is

$$\frac{\Psi}{k}=\frac{213/24}{2}=\frac{213}{48}\approx 4.4.$$

Moreover, if we consider v_1 as the center of the balls, we have that $\delta(v_1,v_2)=1/4$, $\delta(v_1,v_3)=7/36$, $\delta(v_1,v_4)=1/9$, $\delta(v_1,v_5)=3/8$, $\delta(v_1,v_6)=25/63$. In Fig. 4.4 we show the balls centered in v_1 corresponding to $\rho=\delta(v_1,v_4)-\varepsilon$, $\rho=\delta(v_1,v_3)-\varepsilon$, $\rho=\delta(v_1,v_2)-\varepsilon$, $\rho=\delta(v_1,v_5)-\varepsilon$, $\rho=\delta(v_1,v_6)-\varepsilon$, and $\rho=1-\varepsilon$, where ε is abitrarily small. Let us now consider, for example, $\mathcal{B}_\delta(v_1,1/4-\varepsilon)$. This ball contains vertices v_1, v_3, and v_4 and we have that $\mathcal{E}_\delta(v_1,1/4-\varepsilon)=\{(v_1,v_3),(v_1,v_4),(v_3,v_4)\}$ and that $\overline{\mathcal{E}}_\delta(v_1,1/4-\varepsilon)=\{(v_1,v_2),(v_3,v_6),(v_4,v_6)\}$. Hence, it results

$$\begin{aligned}V_\delta(v_1,\frac{1}{4}-\varepsilon) &= \frac{\Psi}{k}+\frac{4}{3}+\frac{4}{9}+\frac{1}{2}+3(\frac{1}{4}-\varepsilon)+2(\frac{1}{4}-\varepsilon-\frac{7}{36})+7(\frac{1}{4}-\varepsilon-\frac{1}{9})\\ &= \frac{213}{48}+16.8+\frac{10}{9}-3\varepsilon\approx 22.31\end{aligned}$$

and

$$C_\delta(v_1,1/4-\varepsilon)=3+2+7=12.$$

Observe that we can consider the ratio $C_\delta(v,\rho)/V_\delta(v,\rho)$ as a function of ρ. Let $\{u_1,u_2,\ldots,u_{|\overline{V}_r|-1}\}$ be the set of nodes in $\overline{V}_r - \{v\}$ ordered by non decreasing distance from v (that is, $i < j \Rightarrow \delta(v,u_i) \leq \delta(v,u_j)$) and let $\{r_1,r_2,\ldots,r_{|\overline{V}_r|-1}\}$ be the corresponding distances from v. It is easy to see that the function $C_\delta(v,\rho)/V_\delta(v,\rho)$ is continuous and non-increasing, for $0 < \rho < r_1$ and for $r_i < \rho < r_{i+1}$ $(i = 1,\ldots,|\overline{V}_r| - 2)$. This implies that it tends to its minimum as it approaches some r_j from below. Notice, moreover, that the derivative of $C_\delta(v,\rho)/V_\delta(v,\rho)$ with respect to ρ is well defined for all values $\rho \notin \{r_1,\ldots,r_{|\overline{V}_r|-1}\}$ (see Fig. 4.3).

Example 4.4 ► Let us consider again the graph shown in Fig. 4.3 and assume that v_1 is the center of the balls. In this case, $u_1 = v_4$, $u_2 = v_3$, $u_3 = v_2$, $u_4 = v_5$, and $u_5 = v_6$. Moreover, $r_1 = 1/9$, $r_2 = 7/36$, $r_3 = 1/4$, $r_4 = 3/8$, and $r_5 = 25/63$. In Fig. 4.5 the behavior of function $C_\delta(v,\rho)/V_\delta(v,\rho)$ is shown in the interval of interest (that is, $(0,25/63)$).

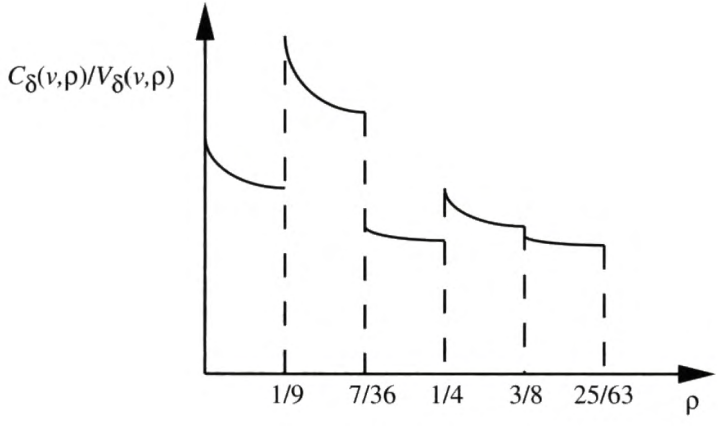


Figure 4.5
The function $C_\delta(v,\rho)/V_\delta(v,\rho)$

Let us now prove that there exists a value $\rho_r < 1/2$ for which the corresponding value $C_\delta(v,\rho_r)/V_\delta(v,\rho_r)$ is sufficiently small.

Lemma 4.4 ► *There exists a value* $\rho_r < 1/2$ *such that*

$$\frac{C_\delta(v,\rho_r)}{V_\delta(v,\rho_r)} \leq 2\ln 2k.$$

PROOF Let us first notice that $C_\delta(v,\rho)$ is the derivative of $V_\delta(v,\rho)$ with respect to ρ. As a consequence, whenever such a derivative is defined, we have

$$\frac{C_\delta(v,\rho)}{V_\delta(v,\rho)} = \frac{\partial}{\partial\rho}(\ln V_\delta(v,\rho)).$$

Assume now, by contradiction, that, for every $\rho \in (0,1/2)$, we have $C_\delta(v,\rho)/V_\delta(v,\rho) > 2\ln 2k$. Two cases are possible:

1. $r_1 \geq 1/2$. This implies that $C_\delta(v,\rho)/V_\delta(v,\rho)$ is differentiable in $(0,1/2)$. As a consequence, we have that

$$\frac{\partial}{\partial \rho}(\ln V_\delta(v,\rho)) > 2\ln 2k$$

and, by integrating both sides in $(0,1/2)$,

$$\ln \frac{V_\delta(v,1/2)}{V_\delta(v,0)} > \ln 2k,$$

which implies $V_\delta(v,1/2) > 2kV_\delta(v,0) = 2\Psi$. This contradicts the fact that $\Psi + \Psi/k$ is the maximum possible value of $V_\delta(v,\rho)$, which is obtained when $\rho \geq r_{|\overline{V}_r|-1}$, i.e., when all the graph is included.

2. There exists t, with $1 \leq t \leq |\overline{V}_r| - 1$, such that $r_1, \ldots, r_t \in (0,1/2)$ (therefore, $C_\delta(v,\rho)/V_\delta(v,\rho)$ might not be differentiable in $(0,1/2)$). Let $r_0 = 0$ and $r_{t+1} = 1/2$. If we apply the same considerations above to each interval (r_i, r_{i+1}), for $i = 0, \ldots, t$, we obtain that

$$\ln V_\delta(v,r_{i+1}) - \ln V_\delta(v,r_i) > (r_{i+1} - r_i)2\ln 2k.$$

Since the volume does not decrease in correspondence to each vertex, by summing over all intervals we get

$$\ln \frac{V_\delta(v,1/2)}{V_\delta(v,0)} = \ln V_\delta(v,1/2) - \ln V_\delta(v,0) > \ln 2k,$$

thus reducing to the previous case.

The lemma is thus proved. QED

We are now ready to prove the main result of this section.

Given an instance x of MINIMUM MULTI-CUT, *the solution returned by Program 4.3 with input x is a feasible solution whose cost $m_{MC}(x)$ satisfies the inequality* ◀ Theorem 4.5

$$m_{MC}(x) < 4\ln(2k)m^*(x).$$

PROOF

Let us first observe that the solution returned by the algorithm is feasible. Indeed, since distances $d(u,v)$ are derived by solving MINIMUM FRACTIONAL MULTI-CUT, we have that $d(s_i,t_i) \geq 1$, for $i = 1, \ldots, k$. As we may observe, for all $i = 1, \ldots, k$, Program 4.3 separates s_i from all vertices at distance at least 1/2: in particular, it separates s_i from t_i. Hence, the solution returned by the algorithm is a multi-cut.

Program 4.3: Multi-cut

input Graph $G = (V, E)$, pairs (s_i, t_i), functions c, d;
output Multi-cut E';
begin
  $r := 1$;
  $\overline{V}_r := V$;
  while $\overline{V}_r \neq \emptyset$ **do begin**
    if there exists i such that $s_i \in \overline{V}_r \wedge t_i \in \overline{V}_r$ **then**
    begin
      Let ρ_r be such that $C_\delta(s_i, \rho_r)/V_\delta(s_i, \rho_r) \leq C_\delta(s_i, \rho)/V_\delta(s_i, \rho)$
        for all $\rho = \delta(s_i, u) - \varepsilon, u \in \overline{V}$ and $\rho < 1/2$;
      $V_r := \mathcal{B}_\delta(s_i, \rho_r)$;
      $r := r + 1$
    $\overline{V}_r := \overline{V}_{r-1} - V_{r-1}$;
    end
    else
      $\overline{V}_r := \emptyset$
  end;
  $E' :=$ set of edges between vertices in V_i and vertices in V_j, for $i \neq j$;
  return E'
end.

For what concerns the performance ratio, let us now prove that $m_{MC}(x) < \ln(2k)\Psi$. The theorem will then follow since Ψ is a lower bound on the optimal measure of MINIMUM MULTI-CUT.

Let $V_1, V_2, \ldots, V_h$ ($h \leq k$) be the subsets of V produced by Program 4.3. Let $V_\delta(s_{i_1}, \rho_1), V_\delta(s_{i_2}, \rho_2), \ldots, V_\delta(s_{i_h}, \rho_h)$ be the volumes of the corresponding balls and let $C_\delta(s_{i_1}, \rho_1), C_\delta(s_{i_2}, \rho_2), \ldots, C_\delta(s_{i_h}, \rho_h)$ be the costs of the corresponding cuts.

By Lemma 4.4, the cost of the multi-cut returned by the algorithm is

$$m_{MC}(x) = \sum_{j=1}^{h} C_\delta(s_{i_j}, \rho_j) \leq 2\ln(2k) \sum_{j=1}^{h} V_\delta(s_{i_j}, \rho_j).$$

Notice now that $\sum_{j=1}^{h} V_\delta(s_{i_j}, \rho_j)$ is equal to the overall volume Ψ of the system, plus $h \leq k$ contributions of size Ψ/k. As a consequence,

$$m_{MC}(x) \leq 2\ln(2k) \sum_{j=1}^{h} V_\delta(s_{i_j}, \rho_j) \leq 4\ln(2k)\Psi,$$

QED and the theorem is proved.

4.2 Between APX and PTAS

AN ASYMPTOTIC approximation scheme is a weaker form of approximation scheme based on the idea that the performance ratio of the returned solution may improve as the optimal measure increases. As we will see in this section, NPO problems exist for which no PTAS can be developed (unless P = NP) but that admit an asymptotic approximation scheme. Let us start with the formal definition of this latter notion.

◀ Definition 4.1
Asymptotic approximation scheme

An NPO *problem* $\mathcal{P}$ *admits an* asymptotic approximation scheme *if there exist an algorithm* $\mathcal{A}$ *and a constant* k *such that, for any instance* x *of* $\mathcal{P}$ *and for any rational* $r \geq 1$, $\mathcal{A}(x,r)$ *returns a solution whose performance ratio is at most* $r + k/m^*(x)$. *Moreover, for any fixed* r, *the running time of* $\mathcal{A}$ *is polynomial.*

The class PTAS^∞ is the set of all NPO problems that admit an asymptotic approximation scheme. Clearly,

$$\text{PTAS} \subseteq \text{PTAS}^\infty \subseteq \text{APX}.$$

Moreover, it is possible to prove that these inclusions are strict if and only if P $\neq$ NP (see Exercise 4.2).

4.2.1 Approximating the edge coloring problem

In this section we consider MINIMUM EDGE COLORING, that is, the problem of coloring the edges of a graph with the minimum number of colors (see Problem 4.2.1).

Minimum Edge Coloring

INSTANCE: Graph $G = (V,E)$.

SOLUTION: A coloring of E, i.e., a partition of E into disjoint sets $E_1, E_2, \ldots, E_k$ such that, for $1 \leq i \leq k$, no two edges in E_i share a common endpoint in G.

MEASURE: The number of colors, i.e., k.

Given a graph G whose maximum degree is Δ, it is easy to see that the optimal value of MINIMUM EDGE COLORING is greater than or equal to Δ. The next result will show that, on the other hand, this value is never greater than $\Delta + 1$. Deciding which one of the two cases holds is an NP-complete problem (see Bibliographical notes) thus implying that MINIMUM EDGE COLORING is not in PTAS (see Exercise 4.9).

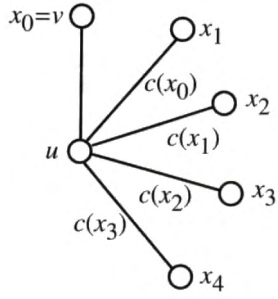


Figure 4.6
The star starting from (u,v)

Theorem 4.6 ▶ *There exists a polynomial-time algorithm $\mathcal{A}$ such that, for any graph G of maximum degree Δ, $\mathcal{A}(G)$ returns an edge-coloring with at most $\Delta+1$ colors.*

PROOF Let $G=(V,E)$ be an instance of MINIMUM EDGE COLORING and let Δ denote the maximum degree of G. The algorithm starts from a new graph $G'=(V,E'=\emptyset)$ (which can be trivially edge-colored with 0 colors) and repeatedly performs the following operations until all edges of E have been *moved* to G':

1. Consider an uncolored edge $(u,v)\notin E'$.
2. Extend the edge-coloring of $G'=(V,E')$ into an edge-coloring of $G''=(V,E'\cup\{(u,v)\})$ with at most $\Delta+1$ colors.
3. Delete (u,v) from E and add it to E'.

We now present step 2 above in detail. Let us assume that a (partial) edge-coloring of G' with at most $\Delta+1$ colors is available such that all edges are colored but edge (u,v). For any node v, in the following we denote by $\mu(v)$ the set of colors that are not used to color edges incident to v. Clearly, if the coloring uses $\Delta+1$ colors, then, for any v, $\mu(v)\neq\emptyset$. In this case, we denote by $c(v)$ one of the colors in $\mu(v)$ (for instance, the color with minimum index).

We can easily find in linear time a sequence $(u,x_0),\ldots,(u,x_s)$ of distinct edges of G' incident to u such that (see Fig. 4.6 where $s=4$):

1. $x_0=v$.
2. For any i with $1\leq i\leq s$, the color of edge (u,x_i) is $c(x_{i-1})$, that is, edge (u,x_i) is colored with a color not used for any edge incident to x_{i-1}.

3. The sequence is maximal, i.e., there is no other edge (u,w) which does not belong to the sequence such that its color is $c(x_s)$.

Note that if $s = 1$, that is, the sequence contains only (u,v), then $\mu(u) \cap \mu(v) \neq \emptyset$: in this case, coloring (u,v) is a trivial task.

We distinguish the following two cases:

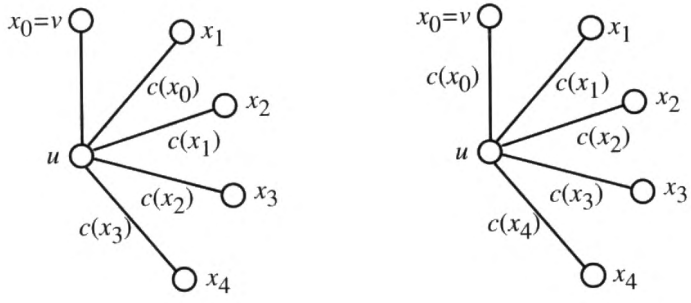


Figure 4.7
The case in which $c(x_4) \in \mu(u)$

1. $c(x_s) \in \mu(u)$: in this case, we can extend the coloring of G' in order to include (u,v) by *shifting* the colors of the sequence. More formally, we start coloring (u,x_s) with $c(x_s)$; then, $c(x_{s-1})$ (that is, the previous color of (u,x_s)) is now in $\mu(u)$, that is, $c(x_{s-1}) \in \mu(u)$. By repeatedly applying this reasoning, we modify the colors of all edges in the sequence but the first one. At the end of this process, we have that $c(v) \in \mu(u)$ so that (u,v) can be colored with $c(v)$ (see Fig. 4.7).

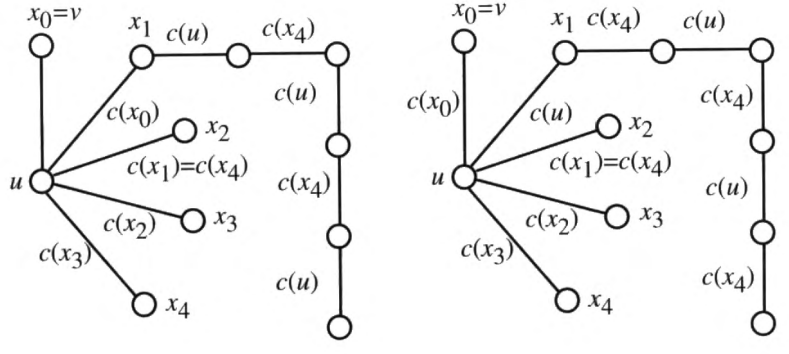


Figure 4.8
The case in which $c(x_4) \notin \mu(u)$ and P_1 does not end in u

2. $c(x_s) \notin \mu(u)$: in this case, one edge (u,x_i) must have been colored with $c(x_s)$ (otherwise, the sequence is not maximal). This implies that $c(x_{i-1}) = c(x_s)$. Let P_{i-1} be the maximal path starting from x_{i-1}

formed by edges whose colors are, alternatively, $c(u)$ and $c(x_s)$ and let w be the last node of this path. Notice that P_{i-1} may have length 0: in such a case, clearly, $w = x_{i-1}$. Two cases may arise.

(a) $w \neq u$: in this case we can interchange colors $c(u)$ and $c(x_s)$ in P_{i-1} and assign color $c(u)$ to (u, x_{i-1}). Note that in this way $c(x_{i-1}) \in \mu(u)$ and, thus, the subsequence preceeding x_{i-1} can be dealed as in Case 1 above (see Fig. 4.8).

(b) $w = u$: in this case, we derive in linear time the path P_s starting from x_s formed by edges whose colors are, alternatively, $c(u)$ and $c(x_s)$. This path cannot intersect P_{i-1}. On the contrary, let z be the first node in P_s that belongs to P_{i-1}. If $z = u$ then there are two edges incident in u with color $c(x_s)$ contradicting the property of a feasible edge coloring of G'. If $z \neq u$, then there must exist three edges incident to z colored with $c(u)$ or $c(x_s)$: once again this contradicts the property of a coloring (see Fig. 4.9).

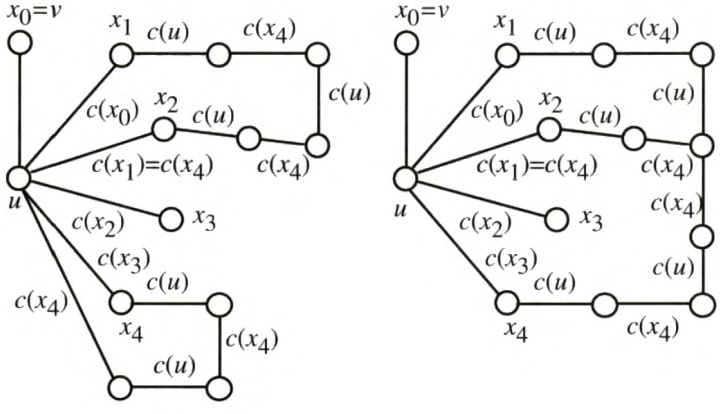


Figure 4.9
P_1 cannot intersect P_4

Since P_s does not intersect P_{i-1}, it cannot end in u: thus, analogously to Case 2.1 above, we can interchange colors $c(u)$ and $c(x_s)$ in P_s and assign color $c(u)$ to (u, x_s). Finally, the subsequence preceeding x_s can be dealed as in Case 1 above (see Fig. 4.10).

In both cases, we have obtained a valid edge-coloring of G' with at most $\Delta + 1$ colors. Since the updating of the coloring of G' has to be done $|E|$ times, it also follows that the running time of the algorithm is bounded by a polynomial.

QED

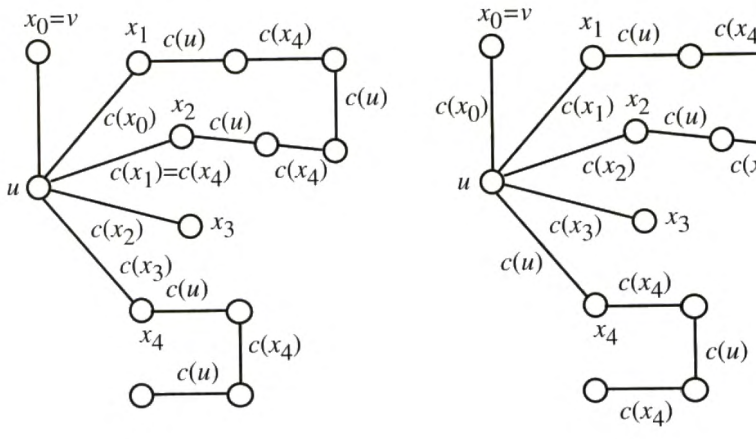


Figure 4.10
The interchange of colors in P_4 and the shift in the remaining sequence

The above theorem implies that a polynomial-time algorithm exists such that, for any graph G, the performance ratio of the solution returned by the algorithm is at most

$$\frac{m^*(G)+1}{m^*(G)} = 1 + \frac{1}{m^*(G)},$$

that is, MINIMUM EDGE COLORING admits an asymptotic approximation scheme with $k = 1$ (observe that a similar argument applies to any NPO problem that admits a polynomial-time algorithm whose absolute error is bounded by a constant). In conclusion, we have the following result.

MINIMUM EDGE COLORING *belongs to the class* PTAS$^\infty$. ◀ Theorem 4.7

4.2.2 Approximating the bin packing problem

In this section we will prove that MINIMUM BIN PACKING belongs to PTAS$^\infty$: note that, in this case, no polynomial-time algorithm whose absolute error is bounded by a constant exists (unless P=NP). We will provide an asymptotic approximation scheme for MINIMUM BIN PACKING, which is based on a combination of the First Fit algorithm and a partitioning technique and is structured into the following five steps.

1. Eliminate *small* items.
2. Group the remaining items into a constant number of size values.
3. Find an optimal solution of the resulting instance.

4. Ungroup the items.
5. Reinsert *small* items.

We will first describe step 3, then steps 1 and 2 along with their "inverse" steps 5 and 4, respectively.

Solving instances of restricted bin packing

For any integer constant $c > 0$ and for any rational constant $\delta \leq 1$, let us consider a restricted version of MINIMUM BIN PACKING in which, for any instance, there are at most c different sizes for the items (i.e., the cardinality of the range of the size function s is bounded by c) and the size of each item is at least $B \cdot \delta$, where B denotes the size of the bins. For the sake of clarity, we will denote this restriction as MINIMUM (c,δ)-RESTRICTED BIN PACKING. Observe that an instance of this problem can be described by the size B of the bins and a multiset $I = \{s_1 : n_1, s_2 : n_2, \ldots, s_c : n_c\}$ in which, the pair $s_i : n_i$, with $1 \leq i \leq c$, denotes the number n_i of items having size s_i.

Example 4.5 ▶ Let us consider the following instance of MINIMUM $(3,3/8)$-RESTRICTED BIN PACKING:

$$I = \{3:4, 5:2, 7:1\} \quad \text{and} \quad B = 8.$$

In this case we have four items with size 3, two with size 5, and one with size 7.

We now show how an instance of MINIMUM (c,δ)-RESTRICTED BIN PACKING can be solved in time $O(n^q)$ where n denotes the number of items and q depends on c and δ only. Equivalently, we show that, for any two constants c and δ, MINIMUM (c,δ)-RESTRICTED BIN PACKING is solvable in polynomial-time.

Let (I,B) be an instance of MINIMUM (c,δ)-RESTRICTED BIN PACKING. The *type* of a bin is a c-vector $\vec{t} = (t_1, \ldots, t_c)$ of integers with $0 \leq t_i \leq n_i$ such that $\sum_{i=1}^{c} t_i s_i \leq B$. In other words, the type specifies for each element s_i of the range of s the number of items having size s_i that are contained in the bin. Observe that for each type $\vec{t}$,

$$\sum_{i=1}^{c} t_i \leq \frac{1}{\delta} \sum_{i=1}^{c} t_i \frac{s_i}{B} \leq \frac{1}{\delta}.$$

This implies that the number of distinct bin types is bounded by the number of ways of choosing c integers whose sum is less than $\lfloor 1/\delta \rfloor$. It is easy to prove that such a number is equal to

$$q = \binom{c + \lfloor \frac{1}{\delta} \rfloor}{\lfloor \frac{1}{\delta} \rfloor}$$

(see Exercise 3.10). Therefore, in any instance of MINIMUM (c,δ)-RESTRICTED BIN PACKING, the number of possible bin types depends on c and δ and does not depend on n.

◀ Example 4.6

Let us consider the instance of Example 4.5. In this case we have that $c = 3$ and $\lfloor 1/\delta \rfloor = 2$. Then

$$q = \binom{5}{2} = 10.$$

Indeed, the possible bin types are only six (i.e., $(0,0,0)$, $(0,0,1)$, $(0,1,0)$, $(1,0,0)$, $(1,1,0)$, and $(2,0,0)$) since the other four bin types (i.e., $(0,0,2)$, $(0,1,1)$, $(0,2,0)$, and $(1,0,1)$) are not feasible because they violate the bound given by the size of the bin.

Since there are at most q different bin types, a feasible solution can now be described by a q-vector $\vec{y} = (y_1, \ldots, y_q)$ where y_i for $i = 1, \ldots, n$ specifies, for each bin type, the number of bins of that type (observe that $0 \leq y_i \leq n$).

◀ Example 4.7

Let us consider again the instance of Example 4.5. A feasible solution can be the one using two bins of type $(2,0,0)$, two bins of type $(0,1,0)$, and one bin of type $(0,0,1)$. One optimal solution, instead, uses two bins of type $(1,1,0)$, one bin of type $(2,0,0)$, and one bin of type $(0,0,1)$.

It is clear that the number of feasible solutions is bounded by $O(n^q)$, which implies that the instance can be solved in time $O(n^q p(n))$ where p is a polynomial by exaustively generating all these feasible solutions (see Exercise 4.10).

Grouping and ungrouping items

Given an instance x of MINIMUM BIN PACKING, let us assume that the n items are ordered according to their size values so that

$$s(u_1) \geq s(u_2) \geq \cdots \geq s(u_n).$$

For any integer $k \leq n$, let $m = \lfloor n/k \rfloor$ and partition the n items into $m+1$ groups G_i with $G_i = \{u_{(i-1)k+1}, \ldots, u_{ik}\}$ for $i = 1, \ldots, m$ and $G_{m+1} = \{u_{mk+1}, \ldots, u_n\}$.

We then define a new instance x_g of MINIMUM BIN PACKING with the same bin size B that, for $i = 2, 3, \ldots, m+1$, contains an item of size $s(u_{(i-1)k+1})$ for each item of instance x that belongs to G_i (that is, the size of all items in the ith group are made equal to that of the largest item in G_i). Note that there at most mk items in x_g.

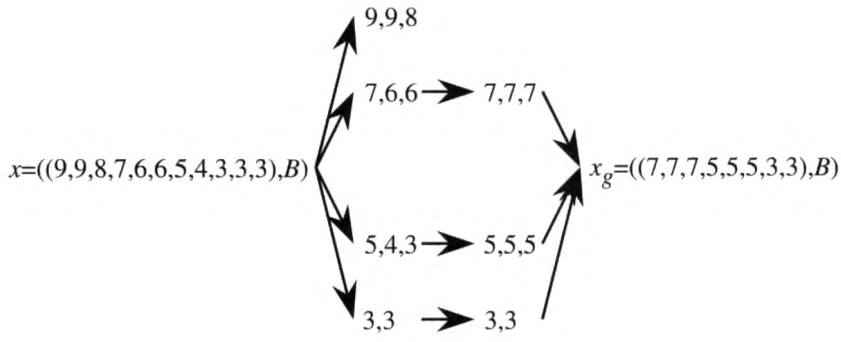


Figure 4.11
An example of grouping items

Example 4.8 ▶ Let us consider the instance x formed by 11 items whose size values are 9, 9, 8, 7, 6, 6, 5, 4, 3, 3, and 3, respectively, and let $k = 3$. We have four groups: $G_1 = \{u_1, u_2, u_3\}$, $G_2 = \{u_4, u_5, u_6\}$, $G_3 = \{u_7, u_8, u_9\}$, and $G_4 = \{u_{10}, u_{11}\}$. The instance x_g contains eight items (see Fig. 4.11): three items of size 7 (corresponding to the items in G_2), three items of size 5 (corresponding to the items in G_3), and two items of size 3 (corresponding to the items in G_4).

Clearly, any packing for the items in x can be transformed into a packing for the items in x_g with the same number of bins by eliminating items in the last group and then substituting each item u in G_i with an item of x_g whose size is equal to $s(u_{ik+1})$ (that is, less than or equal to $s(u)$). Moreover, given a packing of the items in x_g we can obtain a packing of the items in x by simply adding k bins in which we can put the first k items. This implies that

$$m^*(x_g) \leq m^*(x) \leq m^*(x_g) + k, \tag{4.3}$$

that is, if we are able to optimally solve x_g, then we can find a solution for x whose absolute error is at most k.

Dealing with small items

Let x be an instance of bin packing and, for any rational constant $\delta \in (0, 1/2]$, let x_δ be the instance obtained by eliminating all items whose size is less than δB. Given a packing of x_δ with M bins, we can use the First Fit approach to reinsert small items. That is, for each of these items, we insert it into the first bin that can contain it: if it does not fit into any of the currently available bins, then a new bin is created.

At the end of the above procedure two cases are possible.

1. No new bin has been created and the packing uses M bins.

2. $M' \geq 1$ new bins have been created. In this case, similarly to the analysis made in Sect. 2.2.2, we can show that all bins except at most one have an empty space that is at most δB. This implies that

$$(1-\delta)(M+M'-1) \leq \frac{\sum_{i=1}^{n} s(u_i)}{B} \leq m^*(x),$$

that is,

$$M+M' \leq \frac{1}{1-\delta} m^*(x) + 1 \leq (1+2\delta)m^*(x) + 1.$$

In conclusion, given a packing of x_δ with M bins, we can find in polynomial time a solution for x whose measure is at most

$$\max(M, (1+2\delta)m^*(x)+1). \tag{4.4}$$

We have thus completed the description of the five steps of the proposed algorithm for MINIMUM BIN PACKING which is summarized in Program 4.4. Observe that if $r \geq 2$, the First Fit algorithm achieves the desired performance ratio: hence, the input of the algorithm, without loss of generality, is restricted to values of r smaller than 2. We can finally state the main result of this section.

Program 4.4 is an asymptotic polynomial-time approximation scheme for MINIMUM BIN PACKING. ◀ Theorem 4.8

PROOF

To show that, for any fixed $r < 2$, Program 4.4 runs in polynomial time, it suffices to prove that an optimal solution for $x_{\delta,g}$ can be found in polynomial time. Indeed, $x_{\delta,g}$ is an instance of MINIMUM $(\lfloor n'/k \rfloor, \delta)$-RESTRICTED BIN PACKING: since, for any fixed r, both $\lfloor n'/k \rfloor$ and δ are constant, it follows that an optimal solution for $x_{\delta,g}$ can be computed in time $O(n^q p(n))$ where q depends on r and p is a polynomial.

Let us now compute the performance ratio of the packing obtained by the algorithm. To this aim, first observe that the measure of the solution computed by Program 4.4 is $m^*(x_{\delta,g}) + k$. Since all items in x_δ have size at least δB, it follows that $\delta n' \leq m^*(x_\delta)$ and, therefore,

$$k \leq \frac{(r-1)^2}{2} n' + 1 = (r-1)\delta n' + 1 \leq (r-1)m^*(x_\delta) + 1.$$

From Eq. 4.3, it follows that the algorithm packs the items of x_δ into at most

$$m^*(x_{\delta,g}) + k \leq m^*(x_\delta) + (r-1)m^*(x_\delta) + 1 = rm^*(x_\delta) + 1.$$

Program 4.4: Asymptotic Bin Packing

input Instance x of MINIMUM BIN PACKING, rational r with $1 < r < 2$;
output Solution whose measure is at most $rm^*(x) + 1$;
begin
  $\delta := (r-1)/2$;
  Let x_δ be the instance obtained by removing items whose size is less than δB;
  $k := \lceil (r-1)^2 n'/2 \rceil$ where n' is the number of items of x_δ;
  Let $x_{\delta,g}$ be the instance obtained by grouping x_δ with respect to k;
  Find an optimal solution for $x_{\delta,g}$ with measure $m^*(x_{\delta,g})$;
  Insert the first k items of x_δ into k new bins;
  Apply First Fit approach to reinsert the small items;
  return packing obtained
end

Finally, by plugging $r = (1 + 2\delta)$ into Eq. 4.4, we obtain that the total number of used bins is at most

$$\max(rm^*(x_\delta) + 1, rm^*(x) + 1) \leq rm^*(x) + 1,$$

QED which provides the desired performance ratio.

The above theorem states that MINIMUM BIN PACKING belongs to PTAS$^\infty$. It is indeed possible to prove a stronger result that states the existence of an asymptotic approximation scheme whose running time is polynomial both in the length of the input and in $1/(r-1)$ (see Bibliographical notes).

4.3 Exercises

Exercise 4.1 Prove that the analysis of Theorem 4.1 is tight. (Hint: generalize the instance of Example 4.1.)

Exercise 4.2 (*) Prove that if P $\neq$ NP then PTAS $\neq$ PTAS$^\infty$ and PTAS$^\infty \neq$ APX.

Exercise 4.3 Prove that Program 4.5 colors any graph G with n vertices using $O(m^*(G)\log n)$ colors.

Exercise 4.4 Prove that any 2-colorable graph G can be colored in polynomial time using at most 2 colors.

Exercise 4.5 Prove that any graph G of maximum degree $\Delta(G)$ can be colored in polynomial time using at most $\Delta(G) + 1$ colors.

Program 4.5: OptIS Graph Coloring

input A graph $G = (V, E)$;
output A coloring of G with i colors;
begin
  $i := 1$;
  while G is not empty **do**
  begin
    find a maximum independent set S_i in G;
    color vertices of S_i with color i;
    delete from G all vertices in S_i;
    $i := i + 1$
  end
end.

Program 4.6: 3-Coloring

input 3-colorable graph $G = (V, E)$;
output Coloring of vertices of G;
begin
  $H := G$; $n := |V|$;
  $i := 1$;
  while the maximum degree in H is at least $\lceil \sqrt{n} \rceil$ **do**
  begin
    Let v be the vertex of maximum degree in H;
    Let $H_N(v)$ be the graph induced on H by the neighbors of v;
    Color $H_N(v)$ with colors $i, i+1$;
    Color v with color $i+2$;
    $i := i+2$;
    $H :=$ the subgraph of H obtained by deleting v and its neighbors
  end;
  Color all nodes in H with $\Delta(H) + 1$ colors
end.

Exercise 4.6 Prove that, for any 3-colorable graph G, Program 4.6 colors G in polynomial time with at most $3\lceil \sqrt{n} \rceil$ colors. (Hint: use the previous two exercises in order to evaluate the running time.)

Exercise 4.7 (**) Prove that any k-colorable graph G, can be colored in polynomial time with at most $2k\lceil n^{1-1/(k-1)} \rceil$ colors. (Hint: extend the ideas contained in Program 4.6.)

Exercise 4.8 Prove that the two integer linear programming formulations of MINIMUM MULTI-CUT are, indeed, equivalent.

Exercise 4.9 Use the gap technique to show that MINIMUM EDGE COLORING is not in PTAS.

Exercise 4.10 For any fixed c and δ, give a polynomial-time algorithm to solve MINIMUM (c,δ)-RESTRICTED BIN PACKING.

4.4 Bibliographical notes

THE NOTION of input-dependent approximation algorithm is as old as that of constant approximation algorithm: they both appeared in [Johnson, 1974a] which is widely considered as one of the starting points of the systematic study of the complexity of approximation.

The approximation algorithm for MINIMUM SET COVER, which is described in Sect. 4.1.1, was proposed in [Johnson, 1974a] and in [Chvátal, 1979]. This algorithm is optimal: in [Feige, 1996, Raz and Safra, 1997], in fact, it is proved that, for any $\varepsilon > 0$, there is no $(\ln n - \varepsilon)$-approximation algorithm for MINIMUM SET COVER, unless some likely complexity theoretic conjectures fail. It is worth pointing out that it took more than twenty years to close this gap and that MINIMUM SET COVER is one of the very few problems for which optimal approximation algorithms have been proved.

The $O(n/\log n)$-approximate algorithm for MINIMUM GRAPH COLORING is due to [Johnson, 1974a] (from this reference, Exercise 4.3 is also taken): this algorithm is not optimal. Indeed, the best known approximation algorithm for MINIMUM GRAPH COLORING is due to [Halldórsson, 1993a] and has a performance ratio $O(n(\log\log n)^2/(\log n)^3)$. On the other hand, in [Bellare, Goldreich, and Sudan, 1998] it is shown that this problem is not approximable within $n^{1/7-\varepsilon}$ for any $\varepsilon > 0$ unless P = NP. Exercises 4.6 and 4.7 are due to [Wigderson, 1983].

The max-flow min-cut theorem in single-commodity networks was introduced in [Ford and Fulkerson, 1956]. The first corresponding approximation result for the multi-commodity case has been presented in [Leighton and Rao, 1988]. In this paper, the authors consider the related SPARSEST CUT problem, where a cut E' is required on a weighted graph $G = (V, E, c)$ on which k pairs $\{s_i, t_i\} \in V^2$ are defined, such that the ratio $\rho(E') = \sum_{e \in E'} c(e) / \sum_{i \in I(E')} d_i$ is minimum, where $i \in I(E')$ if and only if s_i and t_i are disconnected by the removal of E'. In the same paper an $O(\log n)$ approximation algorithm for this problem is given and a relationship between this problem and the BALANCED CUT is shown, where BALANCED CUT requires, given a value $\alpha \in (0, 1)$, to find a cut of minimum cost which disconnects the graph into two subgraphs each of size at least αn. Such a

problem presents particular relevance as a building block for the design of divide and conquer algorithms on graphs.

The algorithm presented in Sect. 4.1.3 for MINIMUM MULTI-CUT is from [Garg, Vazirani, and Yannakakis, 1996] and has been proved to be applicable to the approximate solution of SPARSEST CUT (and of BALANCED CUT) in [Kahale, 1993]. Other approximation results for SPARSEST CUT has been derived in [Klein, Rao, Agrawal, Ravi, 1995, Plotkin, Tardos, 1995, Linial, London, Rabinovich, 1995, Aumann and Rabani, 1995].

The definition of an asymptotic approximation scheme is taken from [Motwani, 1992] and is based on the notion of asymptotic performance ratio as given in [Garey and Johnson, 1979].

Theorem 4.6 is a fundamental result of graph theory and appeared in [Vizing, 1964]: observe that this theorem also implies that MINIMUM EDGE COLORING is 4/3-approximable. Deciding whether a graph is edge-colorable with Δ colors was shown to be NP-complete in [Holyer, 1981].

The asymptotic approximation scheme for MINIMUM BIN PACKING appeared in [Fernandez de la Vega and Lueker, 1981]: our presentation follows that of [Motwani, 1992]. Indeed, this scheme has been improved in [Karmarkar and Karp, 1982]: in this paper, an asymptotic fully polynomial-time approximation scheme is obtained by means of mathematical programming relaxation techniques and of the ellipsoid method introduced in [Khachian, 1979].