Chapter 3

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 approximation 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 $\mathcal{A}(x) \in \mathrm{SOL}(x)$.*

See Chapter 2, page 39

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:*

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

Deduced from Theorem 2.1

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 $f : V \mapsto \{\text{TRUE}, \text{FALSE}\}$;
begin
  for all $v \in V$ do $f(v) := \text{TRUE}$;
  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 $C_{\bar{l}}$ be the set of clauses in which $\bar{l}$ occurs;
    Let $v_l$ be the variable corresponding to $l$;
    if $l$ is positive then                    (means l = TRUE)
    begin
      $C := C - C_l$;
      Delete $\bar{l}$ from all clauses in $C_{\bar{l}}$; Delete all empty clauses in $C$
    end
    else
    begin
      $f(v_l) := \text{FALSE}$; $C := C - C_{\bar{l}}$;
      Delete $l$ from all clauses in $C_l$; Delete all empty clauses in $C$
    end
  until $C = \emptyset$;
  return $f$
end.
```

Idea of "Greedy Sat":

- If, with the initial assignment f(v)=TRUE, the literal "l" that appears the most takes the value FALSE, then change the involved variable to FALSE, and disregard from further consideration the clauses where "no l" appears and the literals "l" from all other clauses.

-If not, do not change any value, and disregard from further consideration the clauses where "l" appears and the literals "no l" from all other clauses.

-Repeat this until all clauses are disregarded.

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.

Problems 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.

Problems in NPO, but not in APX

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

So, any r-approx. solution will give the optimal solution, with value n.

So, any r-approx. solution will give a measure at least n(1+r)

Corollary 3.4 ▶ *If* P $\neq$ NP, *then* APX $\subset$ NPO. (strictly)

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.

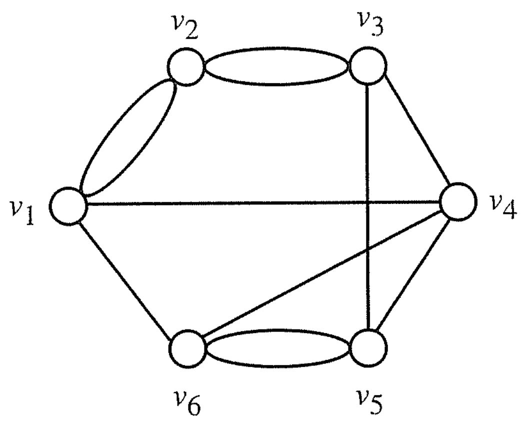


Figure 3.1
An Eulerian multigraph

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 $M = (V, E_T \cup H)$;
   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.

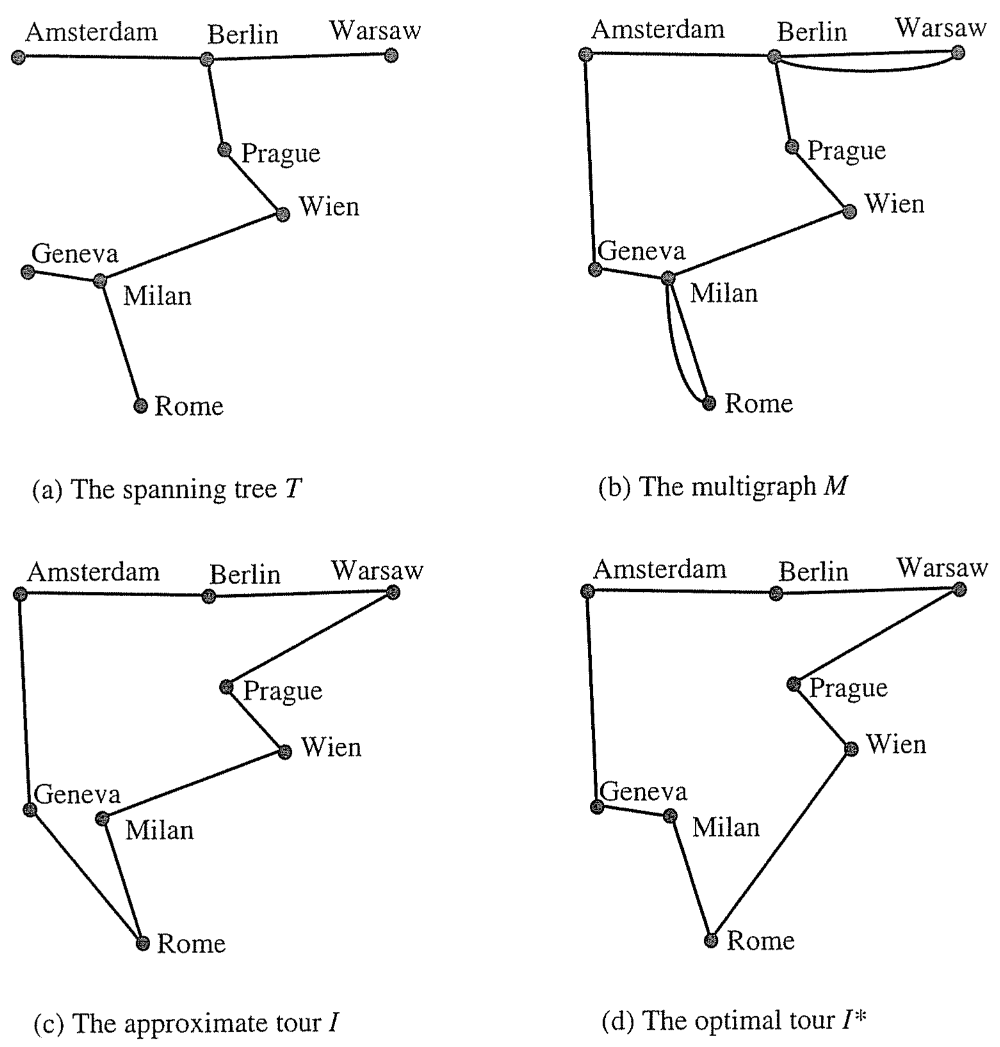
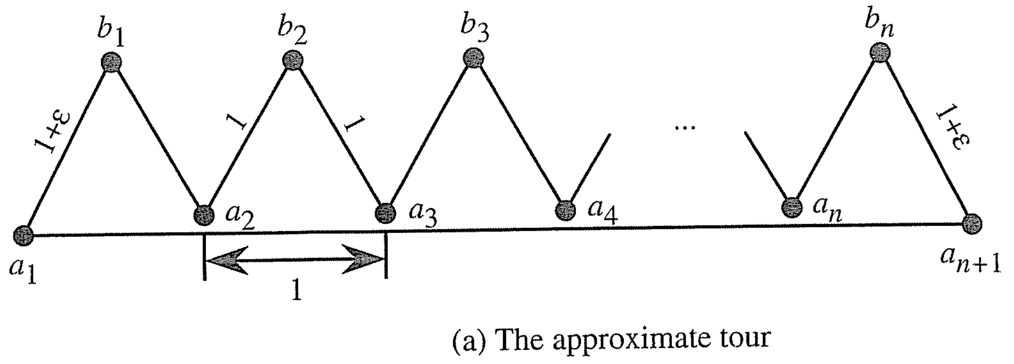

Amsterdam
Berlin
Warsaw
Prague
Wien
Geneva
Milan
Rome
(a) The spanning tree T
(b) The multigraph M
(c) The approximate tour I
(d) The optimal tour I^*

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.



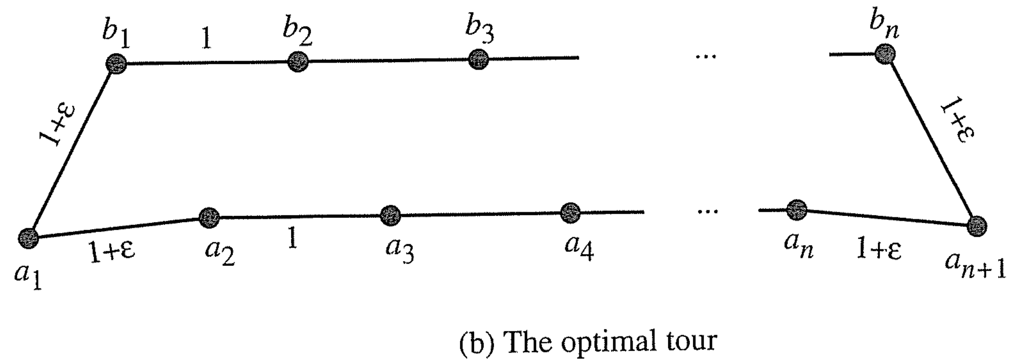


Figure 3.4
Worst case of Christofides' algorithm

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 $c : I_{\mathcal{P}'} \mapsto \mathbb{N}$ 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 $\mathrm{P} = \mathrm{NP}$.

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 $\mathrm{P} = \mathrm{NP}$.

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 $r \geq 2$ then return $X, \emptyset$
   else
   begin
      Sort items in non-increasing order with respect to their weight;
      (*Let $(x_1, \ldots, x_n)$ be the obtained sequence*)
      $k(r) := \lceil (2-r)/(r-1) \rceil$;
      (* First phase *)
      Find an optimal partition $Y_1$, $Y_2$ of $x_1, \ldots, x_{k(r)}$;
      (* Second phase *)
      for $j := k(r)+1$ to $n$ do
         if $\sum_{x_i \in Y_1} a_i \leq \sum_{x_i \in Y_2} a_i$ then
            $Y_1 := Y_1 \cup \{x_j\}$
         else
            $Y_2 := Y_2 \cup \{x_j\}$;
      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.

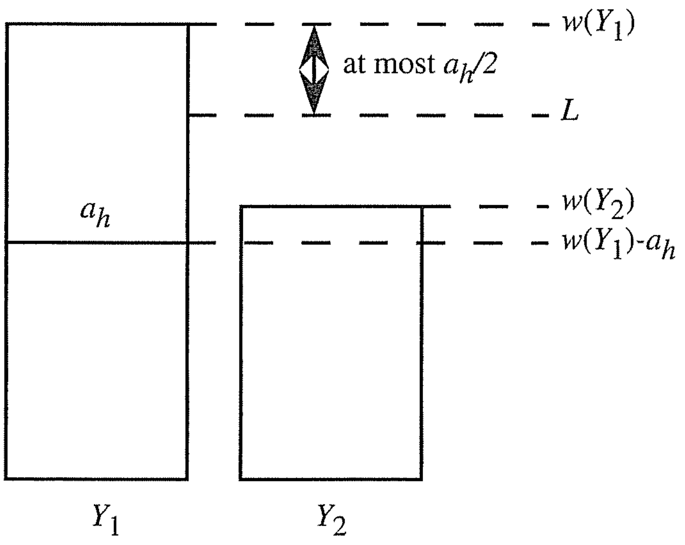


Figure 3.5
The analysis of Program 3.3

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

$$w(Y_1) - L \leq \frac{a_h}{2}. \quad \boxed{1}$$

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

$\boxed{\text{implies}}$ $\boxed{2}$

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 $c : X \mapsto \mathbb{N}$ 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

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

1

2

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.

2^{k(r)} ?

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.

Definition 3.11
Class PTAS

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

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 $d = 1$).

Theorem 3.9

MAXIMUM INTEGER KNAPSACK *belongs to the class* PTAS.

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 $\sum_{x_i \in X} a_i c(x_i) \le b$;
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 $c(x_i) := 0$;
  $F := \{f \mid f : X \mapsto \mathbb{N} \wedge \sum_{i=1}^{n} f(x_i) \le \delta \wedge \sum_{i=1}^{n} a_i f(x_i) \le b\}$;
  for all $f$ in $F$ do
  begin
    $k :=$ maximum $i$ such that $f(x_i) \ne 0$;
    $b_k := b - \sum_{i=1}^{k} a_i f(x_i)$;
    Let $x_{\text{md}}$ be the type of items with maximal value/size ratio in $\{x_k, \ldots, x_n\}$;
    $f(x_{\text{md}}) := f(x_{\text{md}}) + \lfloor b_k / a_{\text{md}} \rfloor$;
    if $\sum_{x_i \in X} p_i f(x_i) \ge \sum_{x_i \in X} p_i c(x_i)$ 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 $c_r(x_{\text{md}}) = b/a_{\text{md}}$ and $c_r(x_i) = 0$ for all other types of items. For any type of item x, let $c(x) = \lfloor c_r(x) \rfloor$: since $m^*(I) \le b p_{\text{md}}/a_{\text{md}}$, we have that function c is a feasible solution of I such that the following holds:

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

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 $\sum_{x \in X} c^*(x) \le \delta$, 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 $\lceil i/3 \rceil$.

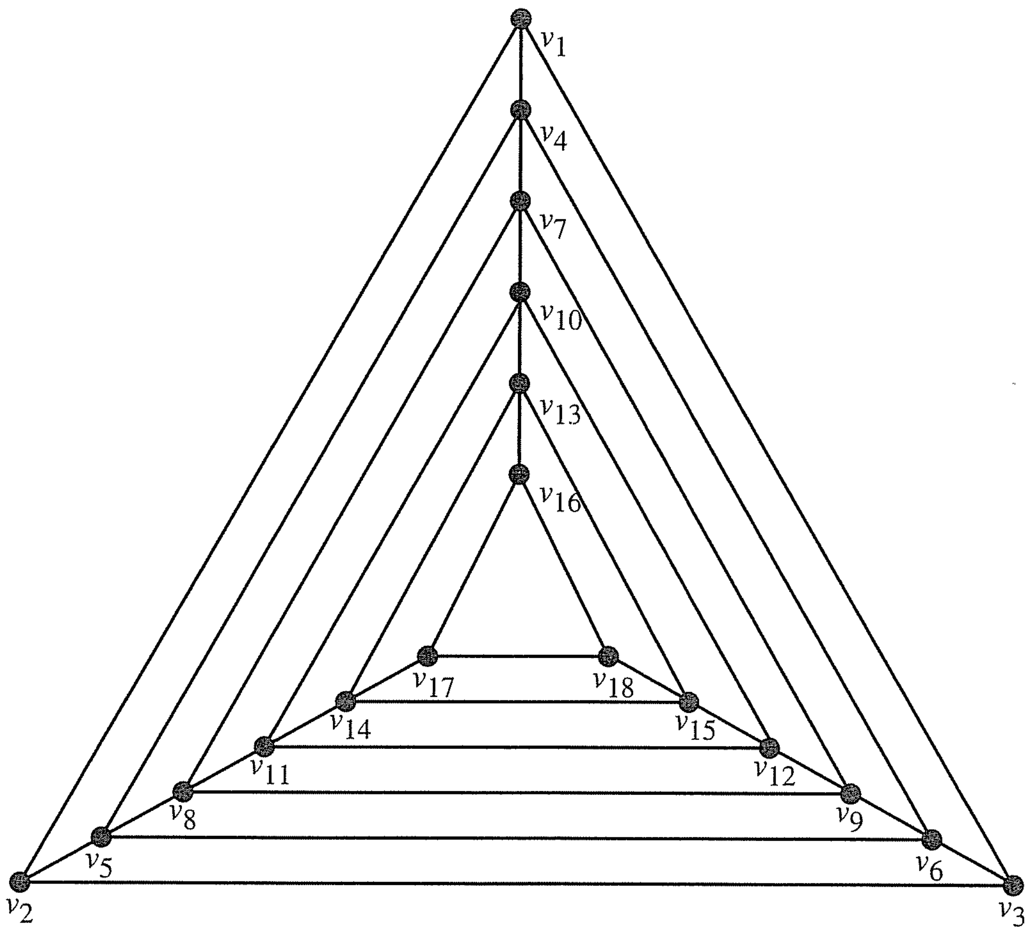


Figure 3.6
An example of a 6-outerplanar embedding

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 $I \subseteq V$;
begin
  $k := \lceil 1/(r-1) \rceil$;
  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 $\overline{V}_i$ be the union of all sets $V_j$ with $j \equiv i \pmod{k+1}$;
      Let $\overline{G}_i$ be the subgraph of $G$ induced by $V - \overline{V}_i$;
      (* $\overline{G}_i$ is $k$-outerplanar *)
      Compute a maximal independent set $I_i$ on $\overline{G}_i$;
    end
  $I := I_m$ such that $|I_m| = \max_{0 \le i \le k}(|I_i|)$;
  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 $0 \le i \le k$, 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 $\overline{G}_i$ 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 $\overline{G}_i$ in time $O(8^k n)$. Let I_m be the maximal cardinality independent set among $\{I_0, \dots, I_k\}$.

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

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.

◀ Theorem 3.11

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

◀ Example 3.6

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 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 $O(r|x|^3/(r-1))$.

See Thm. 2.18

◄ 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.

PO<FPTAS < PTAS < APX < NPO, strictly, unless P=NP

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 satisfies 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.

Theorem 3.14

MAXIMUM PRODUCT KNAPSACK *belongs to the class* FPTAS.

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.

Definition 3.14
Polynomially bounded optimization problem

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

Theorem 3.15
NP-hard: See Def. 1.19

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

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 proble

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.

Definition 3.16 ▶ *Strongly* NP*-hard problem*

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 ≠ 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 ≠ 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).

Recommende page:
http://www.math.ucdavis.edu/~greg/zoology/

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.