The NP-completeness can be shown by providing a polynomial-time Karp-reduction $\mathcal{R}$ from any other problem which is known to be NP-complete. In particular, a polynomial-time Karp-reduction from SATISFIABILITY has been shown in Example 1.10.

As said above, it is not known whether NP-complete problems can be solved efficiently or not. However, due to the fact that even after a wide effort from the whole computer science community, polynomial-time algorithms for such problems (now numbering in the thousands) are still lacking, NP-completeness is considered as a sign that the considered problem is intractable.

1.4 Complexity of optimization problems

LET US now turn our attention to optimization problems. As already observed, most of the basic concepts of complexity theory have been introduced with reference to decision problems. In order to extend the theoretical setting to optimization problems we need to reexamine such concepts and consider how they apply to optimization.

1.4.1 Optimization problems

The study of the cost of solving optimization problems is probably one of the most relevant practical aspects of complexity theory, due to the importance of such problems in many application areas.

In order to extend complexity theory from decision problems to optimization problems, suitable definitions have to be introduced. Also, the relationships between the complexity of optimization problems and the complexity of decision problems have to be discussed.

First, let us provide a formal definition of an optimization problem.

Definition 1.16 ► *Optimization problem*

An optimization problem $\mathcal{P}$ *is characterized by the following quadruple of objects* $(I_{\mathcal{P}}, \mathrm{SOL}_{\mathcal{P}}, m_{\mathcal{P}}, \mathrm{goal}_{\mathcal{P}})$*, where:*

1. $I_{\mathcal{P}}$ *is the set of instances of* $\mathcal{P}$*;*
2. $\mathrm{SOL}_{\mathcal{P}}$ *is a function that associates to any input instance* $x \in I_{\mathcal{P}}$ *the set of* feasible solutions *of* x*;*
3. $m_{\mathcal{P}}$ *is the* measure function*, defined for pairs* (x, y) *such that* $x \in I_{\mathcal{P}}$ *and* $y \in \mathrm{SOL}_{\mathcal{P}}(x)$*. For every such pair* (x, y)*,* $m_{\mathcal{P}}(x, y)$ *provides a*

Problem 1.6: Minimum Vertex Cover

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

SOLUTION: A subset of nodes $U \subseteq V$ such that $\forall (v_i, v_j) \in E$, $v_i \in U$ or $v_j \in U$.

MEASURE: $|U|$.

positive integer which is the value of the feasible solution y;[1]

4. $\text{goal}_{\mathcal{P}} \in \{\text{MIN}, \text{MAX}\}$ *specifies whether* $\mathcal{P}$ *is a maximization or a minimization problem.*

Given an input instance x, we denote by $\text{SOL}^*_{\mathcal{P}}(x)$ the set of *optimal solutions* of x, that is the set of solutions whose value is optimal (minimum or maximum depending on whether $\text{goal}_{\mathcal{P}} = \text{MIN}$ or $\text{goal}_{\mathcal{P}} = \text{MAX}$). More formally, for every $y^*(x)$ such that $y^*(x) \in \text{SOL}^*_{\mathcal{P}}(x)$:

$$m_{\mathcal{P}}(x, y^*(x)) = \text{goal}_{\mathcal{P}}\{v \mid v = m_{\mathcal{P}}(x, z) \wedge z \in \text{SOL}_{\mathcal{P}}(x)\}.$$

The value of any optimal solution $y^*(x)$ of x will be denoted as $m^*_{\mathcal{P}}(x)$. In the following, whenever the problem we are referring to is clear from the context we will not use the subscript that explicitly refers to the problem $\mathcal{P}$.

◀ Example 1.14

Given a graph $G = (V, E)$, the MINIMUM VERTEX COVER problem is to find a vertex cover of minimum size, that is a minimum set U of nodes such that, for each edge $(v_i, v_j) \in E$, either $v_i \in U$ or $v_j \in U$ (that is to say, at least one among v_i and v_j must belong to U). Formally, this problem is defined as follows:

1. $I = \{G = (V, E) \mid G \text{ is a graph}\}$;
2. $\text{SOL}(G) = \{U \subseteq V \mid \forall (v_i, v_j) \in E[v_i \in U \vee v_j \in U]\}$;
3. $m(G, U) = |U|$;
4. $\text{goal} = \text{MIN}$.

In the following, however, whenever a new problem will be introduced, we will make use of the more intuitive notation shown in Problem 1.6.

It is important to notice that any optimization problem $\mathcal{P}$ has an associated decision problem $\mathcal{P}_D$. In the case that $\mathcal{P}$ is a minimization problem, $\mathcal{P}_D$

[1] Notice that, in practice, for several problems the measure function is defined to have values in $\mathbb{Q}$. It is however possible to transform any such optimization problem into an equivalent one satisfying our definition.

Problem 1.7: Vertex cover

INSTANCE: Graph $G = (V, E)$, $K \in \mathbf{N}$.

QUESTION: Does there exist a vertex cover on G of size $\leq K$, that is a subset $U \subseteq V$ such that $|U| \leq K$ and $\forall (u, v) \in E$, $u \in U$ or $v \in U$?

asks, for some $K > 0$, for the existence of a feasible solution y of instance x with value $m(x, y) \leq K$. Analogously, if $\mathcal{P}$ is a maximization problem, the associated decision problem asks, given $K > 0$, for the existence of a feasible solution y of x with $m(x, y) \geq K$. Moreover, an evaluation problem $\mathcal{P}_E$ can also be associated with $\mathcal{P}$, which asks for the value of an optimal solution of $\mathcal{P}$.

More precisely, we can say that the definition of an optimization problem $\mathcal{P}$ naturally leads to the following three different problems, corresponding to different ways of approaching its solution.

Constructive Problem ($\mathcal{P}_C$) – Given an instance $x \in I$, derive an optimal solution $y^*(x) \in \text{SOL}^*(x)$ and its measure $m^*(x)$.

Evaluation Problem ($\mathcal{P}_E$) – Given an instance $x \in I$, derive the value $m^*(x)$.

Decision Problem ($\mathcal{P}_D$) – Given an instance $x \in I$ and a positive integer $K \in \mathbf{Z}^+$, decide whether $m^*(x) \geq K$ (if goal = MAX) or whether $m^*(x) \leq K$ (if goal = MIN). If goal = MAX, the set $\{(x, K) \mid x \in I \wedge m^*(x) \geq K\}$ (or $\{(x, K) \mid x \in I \wedge m^*(x) \leq K\}$ if goal = MIN) is called the *underlying language* of $\mathcal{P}$.

Example 1.15 ▶ The decision problem relative to MINIMUM VERTEX COVER is Problem 1.7. This problem can be shown to be NP-complete (see Exercise 1.12).

Notice that, for any optimization problem $\mathcal{P}$, the corresponding decision problem $\mathcal{P}_D$ is not harder than the constructive problem $\mathcal{P}_C$. In fact, to answer $\mathcal{P}_D$ on instance x it is sufficient to run some algorithm for $\mathcal{P}_C$, thus obtaining the optimal solution $y^*(x)$ together with its value $m(x, y^*(x))$; then, it is sufficient to check whether $m(x, y^*(x)) \leq K$, in the minimization case, or whether $m(x, y^*(x)) \geq K$, in the maximization case.

Example 1.16 ▶ Let us consider MINIMUM TRAVELING SALESPERSON (TSP), that is, Problem 1.8. An instance of this problem can also be represented by a complete graph $G = (V, E)$ of n vertices with positive weights on the edges (the vertices represent the cities and the weight of an edge is equal to the distance between the corresponding pair of cities). Feasible solutions of the problem are then subsets I of

Problem 1.8: Minimum Traveling Salesperson

INSTANCE: Set of cities $\{c_1, \ldots, c_n\}$, $n \times n$ matrix D of distances in Z^+.

SOLUTION: A (traveling salesperson) tour of all cities, that is, a permutation $\{c_{i_1}, \ldots, c_{i_n}\}$.

MEASURE: $\sum_{k=1}^{n-1} D(i_k, i_{k+1}) + D(i_n, i_1)$.

Problem 1.9: Minimum Graph Coloring

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

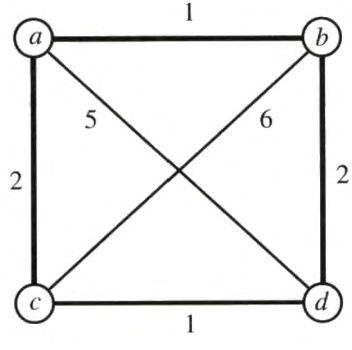
SOLUTION: An assignment $f : V \mapsto \{1, \ldots, K\}$ of K colors to the vertices of G such that $\forall (u, v) \in E,\ f(u) \neq f(v)$.

MEASURE: K.

edges such that the graph (V, I) is a cycle. In Fig. 1.5 it is shown an instance of MINIMUM TRAVELING SALESPERSON in both ways (observe that in this case the distance matrix is symmetric): the thick edges in the graph denote the optimal tour. Since the problem is an optimization problem it cannot belong to NP, but it has a corresponding decision problem that is NP-complete (see Bibliographical notes).

	a	b	c	d
a	0	1	2	5
b	1	0	6	2
c	2	6	0	1
d	5	2	1	0

(a)



(b)

Figure 1.5
An instance of MINIMUM TRAVELING SALESPERSON represented (a) as a matrix and (b) as a graph

◀ Example 1.17

The problem MINIMUM GRAPH COLORING is defined as follows (see Problem 1.9): given a graph $G = (V, E)$, find a vertex coloring with a minimum number of colors, that is a partition of V in a minimum number of classes $\{V_1, \ldots, V_K\}$ such that for any edge $(u, v) \in E$, u and v are in different classes. For example, in the left side of Fig. 1.6 a sample graph is given with a coloring using 4 colors.

Note that such a coloring is not optimal: a minimum coloring for the same graph is shown in the right side of the same figure. As above, the problem cannot belong to NP, but the corresponding decision problem can be shown to be NP-complete (see Exercise 1.14).

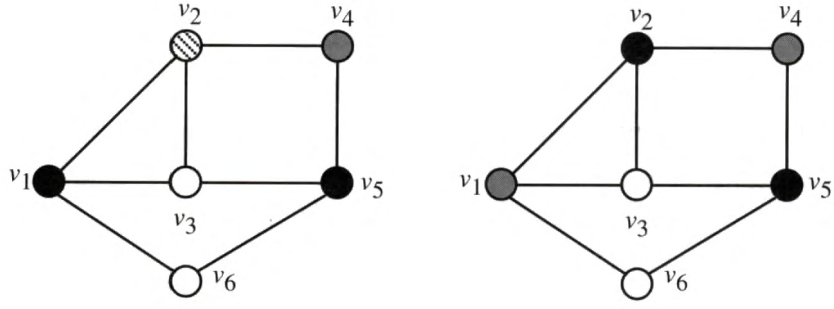


Figure 1.6
A coloring of a graph with 4 colors and an optimal coloring requiring 3 colors

1.4.2 PO and NPO problems

In order to characterize the complexity of optimization problems and to classify such problems accordingly, various approaches may be followed.

The most direct point of view is to consider the time needed for solving the given problem and to extend to optimization problems the theory developed for decision problems.

Of course, the most relevant issue is to characterize optimization problems $\mathcal{P}$ which can be considered tractable, i.e. such that there exists a polynomial-time computable algorithm $\mathcal{A}$ that, for any instance $x \in I$ returns an optimal solution $y \in \text{SOL}^*(x)$, together with its value $m^*(x)$. This means that our main concern will be the study of constructive versions of optimization problems: this will indeed be the approach that will be followed throughout the book (even though we will usually avoid to explicitly return the measure of the computed solution).

Example 1.18 ▶ The problem MINIMUM PATH is defined as follows (see Problem 1.10): given a graph $G = (V, E)$ and a pair of nodes v_s and v_d, derive the shortest path from v_s to v_d. This problem can be proved to be tractable by showing the polynomial-time Program 1.4 which constructs all minimum paths from all nodes in V to v_d. The algorithm visits the graph in a breadth-first order and constructs a tree with root v_d by setting in any node a pointer π to its parent node. An example of the behavior of the algorithm is shown in Fig. 1.7 where the lower part represents the resulting tree obtained with input the graph depicted in the upper part and $v_d = v_1$: the arrows denote the pointers to the parents while the values in the square brackets represent the visiting order.

Problem 1.10: Minimum Path

INSTANCE: Graph $G = (V, E)$, two nodes $v_s, v_d \in V$.

SOLUTION: A path $(v_s = v_{i_1}, v_{i_2}, \ldots, v_{i_k} = v_d)$ from v_s to v_d.

MEASURE: The number k of nodes in the path.

Program 1.4: Shortest path by breadth-first search

```
input Graph G = (V,E) and two nodes v_s, v_d ∈ V;
output Minimum path from v_s to v_d in G;
begin
   Enqueue(v_d, Q);
   Mark node v_d as visited;
   π[v_d] := nil;
   while Q is not empty do
   begin
      v := Dequeue(Q);
      for each node u adjacent to v do
         if u has not been already visited then
         begin
            Enqueue(u, Q);
            Mark u as visited;
            π[u] := v;
         end
   end;
   if v_s has been visited then
      return the path from v_s to v_d by following pointers π;
end.
```

In this book we are mainly interested in those optimization problems which stand on the borderline between tractability and intractability and which, by analogy with the NP decision problems, are called NPO problems.

◀ Definition 1.17
Class NPO

An optimization problem $\mathcal{P} = (I, \mathrm{SOL}, m, \mathrm{goal})$ *belongs to the class* NPO *if the following holds:*

1. *the set of instances* I *is recognizable in polynomial time;*
2. *there exists a polynomial* q *such that, given an instance* $x \in I$*, for any* $y \in \mathrm{SOL}(x)$, $|y| \leq q(|x|)$ *and, besides, for any* y *such that* $|y| \leq q(|x|)$*, it is decidable in polynomial time whether* $y \in \mathrm{SOL}(x)$*;*
3. *the measure function* m *is computable in polynomial time.*

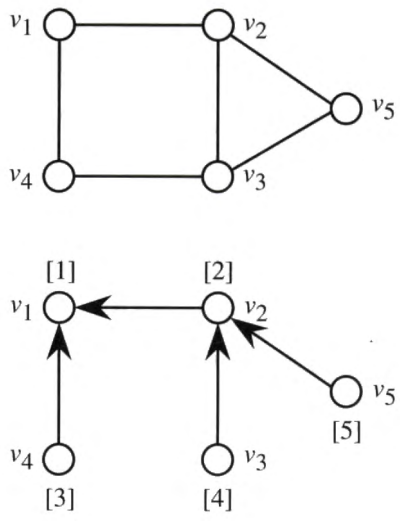


Figure 1.7
An example of application of Program 1.4

Example 1.19 ▶ MINIMUM VERTEX COVER belongs to NPO since:

1. the set of instances (any undirected graph) is clearly recognizable in polynomial time;
2. since any feasible solution is a subset of the set of nodes its size is smaller than the size of the instance; moreover, testing that a subset $U \subseteq V$ is a feasible solution requires testing whether any edge in E is incident to at least one node in U, which can be clearly performed in polynomial time;
3. given a feasible solution U, the measure function (size of U) is trivially computable in polynomial time.

Even if not explicitly introduced, underlying the definition of NPO problem there is a nondeterministic computation model. This is formally stated by the following result which basically shows that the class NPO is the natural optimization counterpart of the class NP.

Theorem 1.1 ▶ *For any optimization problem $\mathcal{P}$ in* NPO, *the corresponding decision problem $\mathcal{P}_D$ belongs to* NP.

PROOF Assume that $\mathcal{P}$ is a maximization problem (the proof in the minimization case is similar). Given an instance $x \in I$ and an integer K, we can solve $\mathcal{P}_D$ by performing the following nondeterministic algorithm. In time $q(|x|)$, where q is a polynomial, any string y such that $|y| \leq q(|x|)$ is guessed. Afterwards the string is tested for membership in $\mathrm{SOL}(x)$ in polynomial time. If the test is positive, $m(x,y)$ is computed (again in polynomial time) and if $m(x,y) \geq K$ the answer YES is returned. Otherwise (i.e., either y is not a feasible solution or $m(x,y) < K$), the answer NO is returned.

QED

The relationship between classes NP and NPO, which holds in the case of nondeterministic computations, can be translated, in the case of deterministic algorithms, by the following definition, that introduces the class of NPO problems whose constructive versions are efficiently solvable .

◀ Definition 1.18
Class PO

An optimization problem $\mathcal{P}$ belongs to the class PO *if it is in* NPO *and there exists a polynomial-time computable algorithm $\mathcal{A}$ that, for any instance $x \in I$, returns an optimal solution $y \in \mathrm{SOL}^*(x)$, together with its value $m^*(x)$.*

◀ Example 1.20

MINIMUM PATH belongs to PO. Indeed, it is easy to see that this problem satisfies the three conditions of Def. 1.17. Moreover, as we have seen in Example 1.18, MINIMUM PATH is solvable in polynomial time.

Practically all interesting optimization problems belong to the class NPO: in addition to all optimization problems in PO (such as MINIMUM PATH), several other problems of great practical relevance belong to NPO. Beside MINIMUM VERTEX COVER, MINIMUM TRAVELING SALESPERSON, and MINIMUM GRAPH COLORING, many other graph optimization problems, most packing and scheduling problems, and the general formulations of integer and binary linear programming belong to NPO but are not known to be in PO since no polynomial-time algorithm for them is known (and it is likely that none exists). For some of them, not only does the exact solution appear to be extremely complex to obtain but, as we will see, even to achieve good approximate solutions may be computationally hard.

Indeed, for all the optimization problems in NPO – PO the intrinsic complexity is not precisely known. Just as with the decision problems in NP – P, no polynomial-time algorithms for them have been found but no proof of intractability is known either. In fact, the question "PO = NPO?" is strictly related to the question "P = NP?" since it can be proved that the two questions are equivalent in the sense that a positive answer to the first would imply a positive answer to the second and vice versa. In order to establish such relationships between the two questions we have to make more precise how the complexity of decision problems may be related to the complexity of optimization problems.

1.4.3 NP-hard optimization problems

In order to assess the intrinsic complexity of optimization problems we may think of proceeding as in the case of decision problems. In that case the relative complexity of problems was established by making use

of polynomial-time Karp reductions and of the related notion of NP-completeness. Unfortunately, Karp reductions are defined for decision problems and cannot be applied to optimization problems. In this case, instead, we can make use of the polynomial-time Turing reducibility (see Sect. 1.3).

Definition 1.19 ▶ *NP-hard problem*

An optimization problem $\mathcal{P}$ is called NP*-hard if, for every decision problem $\mathcal{P}' \in$ NP, $\mathcal{P}' \leq_T^p \mathcal{P}$, that is, $\mathcal{P}'$ can be solved in polynomial time by an algorithm which uses an oracle that, for any instance $x \in I_{\mathcal{P}}$, returns an optimal solution $y^*(x)$ of x along with its value $m^*_{\mathcal{P}}(x)$.*[2]

Thus, a problem is NP-hard if it is at least as difficult to solve (in terms of time complexity and apart from a polynomial-time reduction) as any problem in NP. As a consequence of the definition of NP-completeness, in order to prove that an optimization problem $\mathcal{P}$ is NP-hard it is enough to show that $\mathcal{P}' \leq_T^p \mathcal{P}$ for an NP-complete problem $\mathcal{P}'$. Besides, if a problem $\mathcal{P}$ is NP-hard, $\mathcal{P} \in$ PO implies P=NP.

Indeed, many interesting problems are NP-hard. For example, this happens with all problems in NPO whose underlying language is NP-complete.

Theorem 1.2 ▶ *Let a problem $\mathcal{P} \in$ NPO be given; if the underlying language of $\mathcal{P}$ is* NP*-complete then $\mathcal{P}$ is* NP*-hard.*

PROOF Clearly, the solution of the decision problem could be obtained for free if an oracle would give us the solution of the constructive optimization problem.

QED

From the preceding result we may easily derive a first consequence concerning the complexity of NPO problems. In fact it turns out that if we could solve the problem $\mathcal{P}$ of Theorem 1.2 in polynomial time we could also solve its underlying decision problem in polynomial time. Hence, unless P = NP, no problem in NPO whose underlying language is NP-complete (e.g., MINIMUM TRAVELING SALESPERSON) can belong to PO and the next result follows.

Corollary 1.3 ▶ *If* P $\neq$ NP *then* PO $\neq$ NPO.

The fact that an NPO problem $\mathcal{P}$ is NP-hard naturally places $\mathcal{P}$ at the highest level of complexity in the class NPO, like what happens in NP with the NP-complete problems.

Example 1.21 ▶ As a consequence of Exercises 1.12 and 1.14 and of Example 1.16, we have that MINIMUM VERTEX COVER, MINIMUM GRAPH COLORING, and MINIMUM TRAVELING SALESPERSON are NP-hard.

[2] More formally, and according to the definition of Turing reducibility (i.e., Def. 1.12), we should write $\mathcal{P}' \leq_T^p \text{SOL}^*_{\mathcal{P}}$ instead of $\mathcal{P}' \leq_T^p \mathcal{P}$.

1.4.4 Optimization problems and evaluation problems

As we have seen, the results of the preceding section have provided us with some information on the relative complexity of decision problems and optimization problems in some particular cases.

Let us now see the mutual relations existing among the various versions of optimization problems (decision, evaluation, and constructive problems) in a more systematic manner.

First of all, let us state more formally three results that have already been mentioned in the preceding section.

◀ Theorem 1.4

For any problem $\mathcal{P} \in$ NPO, $\mathcal{P}_D \equiv_T^p \mathcal{P}_E \leq_T^p \mathcal{P}_C$.

PROOF

The proofs of $\mathcal{P}_D \leq_T^p \mathcal{P}_E$ and of $\mathcal{P}_E \leq_T^p \mathcal{P}_C$ are immediate. Regarding $\mathcal{P}_E \leq_T^p \mathcal{P}_D$, due to the properties of NPO problems we have that, for any $x \in I$ and for any $y \in \text{SOL}(x)$, the range of possible values of $m(x,y)$ is bounded by $M = 2^{p(|x|)}$ for some polynomial p. Hence, by applying binary search, the evaluation problem can be solved by at most $\log M = p(|x|)$ queries to the oracle $\mathcal{P}_D$.

QED

Less clear is the possibility of deriving the constructive solution when knowing only the solution of the evaluation problem.

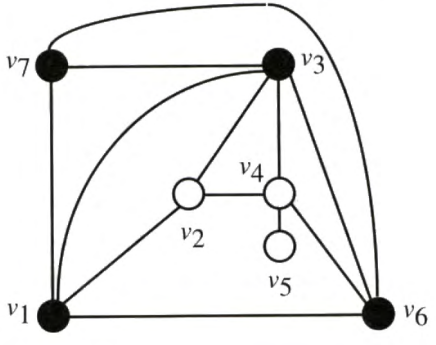


Figure 1.8
A clique of size 4

◀ Example 1.22

Let us consider MAXIMUM CLIQUE, that is, Problem 1.11. An example of a clique of size 4 is shown in Fig. 1.8. Given the solution of the evaluation problem for MAXIMUM CLIQUE we can construct an optimal solution in polynomial time as described by Program 1.5 where, given a node v, $G(v)$ denotes the subgraph induced by v and the set of nodes adjacent to v and $G^-(v)$ denotes the subgraph induced by the set of nodes adjacent to v.

The cost of running Program 1.5 is given by the following recurrence relation as a function T of the number of nodes in the graph:

1. $T(1)$ is $O(1)$,

Problem 1.11: Maximum Clique

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

SOLUTION: A clique in G, i.e., a subset of nodes $U \subseteq V$ such that $\forall (v_i, v_j) \in U \times U, (v_i, v_j) \in E$ or $v_i = v_j$.

MEASURE: $|U|$.

Program 1.5: Maxclique

input Graph $G = (V, E)$;
output Maximum clique in G;
begin
  $k :=$ MAXIMUM CLIQUE$_E(G)$;
  if $k = 1$ **then return** any node in V;
  Find node v such that MAXIMUM CLIQUE$_E(G(v)) = k$;
  return $\{v\} \cup$ Maxclique$(G^-(v))$
end.

2. $T(n) = (n+1) + T(n-1)$

(recall that querying the oracle MAXIMUM CLIQUE$_E$ costs only one step). The solution of the recurrence relation is $O(n^2)$.

Unfortunately, the straightforward construction of the previous example cannot be applied in general. Intuitively it may be that the constructive problem is indeed more complex than the evaluation problem since it yields additional information.

The next result, however, shows that whenever the decision problem is NP-complete the constructive, evaluation, and decision problems are equivalent.

Theorem 1.5 ▶ *Let $\mathcal{P}$ be an* NPO *problem whose underlying language $\mathcal{P}_D$ is* NP*-complete. Then $\mathcal{P}_C \leq_T^p \mathcal{P}_D$.*

PROOF Let us assume, without loss of generality, that $\mathcal{P}$ is a maximization problem. To prove the theorem we will derive an NPO problem $\mathcal{P}'$ such that $\mathcal{P}_C \leq_T^p \mathcal{P}'_D$, and then use the fact that, since $\mathcal{P}_D$ is NP-complete, $\mathcal{P}'_D \leq_T^p \mathcal{P}_D$ (actually, $\mathcal{P}'_D \leq_m^p \mathcal{P}_D$).

Problem $\mathcal{P}'$ has the same definition of $\mathcal{P}$ except for the measure function $m_{\mathcal{P}'}$, which is defined as follows. Let p be a polynomial which bounds the length of the solutions of $\mathcal{P}$ with respect to the length of the corresponding instances and, for any solution $y \in \text{SOL}_{\mathcal{P}}$, let $\lambda(y)$ denote the rank of y

in the lexicographic order. Then, for any instance $x \in I_{\mathcal{P}'} = I_{\mathcal{P}}$ and for any solution $y \in \mathrm{SOL}_{\mathcal{P}'}(x) = \mathrm{SOL}_{\mathcal{P}}(x)$, let $m_{\mathcal{P}'}(x,y) = 2^{p(|x|)+1} m_{\mathcal{P}}(x,y) + \lambda(y)$.

Notice that, for all instances x of $\mathcal{P}'$ and for any pair $y_1, y_2 \in \mathrm{SOL}_{\mathcal{P}'}(x)$, we have that $m_{\mathcal{P}'}(x,y_1) \neq m_{\mathcal{P}'}(x,y_2)$. Therefore there exists a unique optimal feasible solution $y^*_{\mathcal{P}'}(x)$ in $\mathrm{SOL}^*_{\mathcal{P}'}(x)$. Note also that, by definition, if $m_{\mathcal{P}'}(x,y_1) > m_{\mathcal{P}'}(x,y_2)$ then $m_{\mathcal{P}}(x,y_1) \geq m_{\mathcal{P}}(x,y_2)$, thus implying that $y^*_{\mathcal{P}'}(x) \in \mathrm{SOL}^*_{\mathcal{P}}(x)$.

The optimal solution $y^*_{\mathcal{P}'}(x)$ can be easily derived in polynomial time by means of an oracle for $\mathcal{P}'_E$, since, given $m^*_{\mathcal{P}'}(x)$, the position of $y^*_{\mathcal{P}'}(x)$ in the lexicographic order can be derived by computing the remainder of the division between $m^*_{\mathcal{P}'}(x)$ and $2^{p(|x|)+1}$.

We know that an oracle for $\mathcal{P}'_D$ can be used to simulate $\mathcal{P}'_E$ in polynomial time. Thus we can construct an optimal solution of $\mathcal{P}$ in polynomial time using an oracle for $\mathcal{P}'_D$, and since $\mathcal{P}'_D \in$ NP and $\mathcal{P}_D$ is NP-complete, an oracle for $\mathcal{P}_D$ can be used to simulate the oracle for $\mathcal{P}'_D$. QED

The following question still remains open: Is there an NPO problem $\mathcal{P}$ whose corresponding constructive problem is harder than the evaluation problem $\mathcal{P}_E$? Indeed, there is some evidence that the answer to this question may be affirmative (see Bibliographical notes).

1.5 Exercises

Exercise 1.1 Prove the cost evaluations given in Example 1.1.

Exercise 1.2 Program 1.6 is the Euclidean algorithm for the greatest common divisor of two integers $x, y \in \mathbf{Z}$. Determine its execution cost under the uniform cost model and the logarithmic cost model. Moreover, show which are the dominant operations and derive the asymptotic execution cost.

Exercise 1.3 Given a set P of points in the plane, the convex hull of P is defined as the minimal size convex polygon including all points in P. It can be easily proved that the set of vertices of the convex hull is a subset of P. Prove that the time complexity of the problem of computing the convex hull of a set of n points is:

1. $O(n^2)$;
2. $\Omega(n \log n)$ (by reduction from sorting);
3. (*) $\Theta(n \log n)$.

Program 1.6: Euclid

```
input Integers x, y;
output Greatest common divisor z of x and y;
begin
  while x > 0 do
    if x > y then
      x := x − y
    else if x < y then
      y := y − x
    else begin
      z := x;
      x := 0
    end;
  return z
end.
```

Problem 1.12: Maximum Path in a Tree

INSTANCE: Tree T, integer $K > 0$.

QUESTION: Is the length of the longest path in T less than K?

Exercise 1.4 Let us denote as LOGSPACE the class of all problems solvable in *work space* proportional to the logarithm of the input size where the work space is the number of memory locations used for performing the computation, not considering the space required for the initial description of the problem instance (for example, we may assume that the problem instance is represented on a read-only memory device, whose size is not considered in the space complexity evaluation). Prove that SATISFYING TRUTH ASSIGNMENT is in LOGSPACE.

Exercise 1.5 Show that Problem 1.12 is in LOGSPACE.

Exercise 1.6 Derive a polynomial-space algorithm solving QUANTIFIED BOOLEAN FORMULAS.

Exercise 1.7 Show that polynomial-time Karp reductions have the transitive property, that is, if $\mathcal{P}_1 \leq_m^p \mathcal{P}_2$ and $\mathcal{P}_2 \leq_m^p \mathcal{P}_3$, then $\mathcal{P}_1 \leq_m^p \mathcal{P}_3$.

Exercise 1.8 Define a polynomial-time nondeterministic algorithm for the Problem 1.13.

Exercise 1.9 Recall that a *disjunctive normal form* (DNF) formula is defined as a collection of conjunctions $C = \{c_1, c_2, \ldots, c_m\}$ and is assumed to

Problem 1.13: Subset Sum

INSTANCE: Set $S = \{s_1, s_2, \ldots, s_n\}$, weight function $w : S \mapsto \mathbf{N}$, $K \in \mathbf{N}$.

QUESTION: Does there exist any $S' \subseteq S$ such that $\sum_{s_i \in S'} w(s_i) = K$?

Problem 1.14: Tautology

INSTANCE: Boolean formula $\mathcal{F}$ in DNF.

QUESTION: Is it true that every truth assignment on $\mathcal{F}$ is a satisfying assignment?

be satisfied by a truth assignment f if and only if at least one conjunction c_i, with $1 \leq i \leq m$, is satisfied by f. Show that Problem 1.14 is in co–NP.

Exercise 1.10 Show that NP is closed with respect to polynomial-time Karp-reducibility. Is NP closed also with respect to polynomial-time Turing reducibility?

Exercise 1.11 Prove that if there exists a problem $\mathcal{P}$ such that both $\mathcal{P}$ and $\mathcal{P}^c$ are NP-complete then NP = co–NP.

Exercise 1.12 (*) Prove that VERTEX COVER is NP-complete. (Hint: use the NP-completeness of SATISFIABILITY.)

Exercise 1.13 Prove that the decision problem corresponding to the MAXIMUM CLIQUE problem is NP-complete. (Hint: use the previous exercise).

Exercise 1.14 (*) Prove that the decision problem corresponding to the MINIMUM GRAPH COLORING problem is NP-complete. (Hint: use the NP-completeness of SATISFIABILITY.)

Exercise 1.15 Prove that the problem of deciding whether a graph is colorable with two colors is in P. (Hint: assign a color to a vertex and compute the consequences of this assignment.)

Exercise 1.16 Prove that the optimization problem corresponding to Problem 1.15 is in NPO.

Exercise 1.17 A problem $\mathcal{P}$ is called NP-easy if there exists a decision problem $\mathcal{P}' \in$ NP such that $\mathcal{P} \leq_T^p \mathcal{P}'$. Show that MINIMUM TRAVELING SALESPERSON is NP-easy. (Hint: Use the language $\{(G, p, k) \mid G$ is a graph, p is a path that can be extended to a Hamiltonian tour of cost $\leq k\}$ as the problem $\mathcal{P}'$ in the definition.)

Problem 1.15: 4-th vertex cover

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

QUESTION: Is the size of the 4-th smallest vertex cover in G at most equal to K?

Problem 1.16: MAXIMUM SATISFIABILITY

INSTANCE: Set C of disjunctive clauses on a set of variables V.

SOLUTION: A truth assignment $f : V \mapsto \{\text{TRUE}, \text{FALSE}\}$.

MEASURE: The number of clauses in C which are satisfied by f.

Exercise 1.18 Given an oracle for the evaluation version of Problem 1.16, show how it can be exploited to solve the constructive version of the same problem.

1.6 Bibliographical notes

GENERAL INTRODUCTIONS to the theory of algorithms and techniques for their analysis are given in [Knuth, 1969, Knuth, 1971, Knuth, 1973]. Other fundamental references to techniques for the design and analysis of algorithms are [Aho, Hopcroft, and Ullman, 1974] and [Cormen, Leiserson, and Rivest, 1990]. A recent text entirely devoted to formal methods for the analysis of combinatorial algorithms is [Sedgewick and Flajolet, 1996].

A detailed exposition of the theory of computational complexity of decision problems and of structural properties of related complexity classes is provided in several textbooks such as [Balcàzar, Diaz, and Gabarrò, 1988, Bovet and Crescenzi, 1994, Papadimitriou, 1994]. An excellent textbook on the theory of computability is [Rogers, 1987], where a detailed exposition of several types of reducibilities can be found.

The theory of NP-completeness is one of the bases for the topics developed in this textbook: the original concept of NP-complete problems has been introduced independently in [Cook, 1971], where SATISFIABILITY was proved to be NP-complete, and in [Levin, 1973]. The theory was further refined in [Karp, 1972], where a first set of problems were shown to be NP-complete by using reductions.

[Garey and Johnson, 1979] represents a landmark in the literature on NP-completeness and provides a detailed survey of the theory and an extensive

list of NP-complete problems. Still twenty years later, this book is a fundamental reference for people interested in the assessing of the computational tractability or intractability of decision problems. The compendium of optimization problems contained in this book and the notation therein adopted are directly inspired by the list of NP-complete problems given in [Garey and Johnson, 1979]. This book is also a source of information on PSPACE-completeness and on the first examples of PSPACE-complete problems, such as QUANTIFIED BOOLEAN FORMULAS.

The characterization of the complexity of optimization problems has been first addressed in [Johnson, 1974a, Ausiello, Marchetti-Spaccamela, and Protasi, 1980, Paz and Moran, 1981], by establishing connections between combinatorial properties and complexity of decision and optimization problems. In particular, the concept of strong NP-completeness introduced in [Garey and Johnson, 1978] proved to be very fruitful in the theory of approximation of optimization problems (see Chap. 3). In [Krentel, 1988] the first structural approach to the characterization of the complexity of optimization problems is provided, by introducing a suitable computation model and the corresponding complexity classes (see Chap. 6). Further studies on the relationship between decision, evaluation, and optimization problems, including Theorem 1.5, are discussed in [Crescenzi and Silvestri, 1990].