PART I

Complexity of Combinatorial Optimization Problems

Chapter 1

Basic Concepts in Algorithms and Complexity Theory

1.1. Algorithmic complexity

In algorithmic theory, a problem is a general question to which we wish to find an answer. This question usually has parameters or variables the values of which have yet to be determined. A problem is posed by giving a list of these parameters as well as the properties to which the answer must conform. An instance of a problem is obtained by giving explicit values to each of the parameters of the instanced problem.

An algorithm is a sequence of elementary operations (variable affectation, tests, forks, etc.) that, when given an instance of a problem as input, gives the solution of this problem as output after execution of the final operation.

The two most important parameters for measuring the quality of an algorithm are: its *execution time* and the *memory space* that it uses. The first parameter is expressed in terms of the number of instructions necessary to run the algorithm. The use of the number of instructions as a unit of time is justified by the fact that the same program will use the same number of instructions on two different machines but the time taken will vary, depending on the respective speeds of the machines. We generally consider that an instruction equates to an elementary operation, for example an assignment, a test, an addition, a multiplication, a trace, etc. What we call the *complexity in time* or simply the *complexity* of an algorithm gives us an indication of the time it will take to solve a problem of a given size. In reality this is a function that associates an order of

---

Chapter written by Vangelis Th. PASCHOS.

magnitude[1] of the number of instructions necessary for the solution of a given problem with the size of an instance of that problem. The second parameter corresponds to the number of memory units used by the algorithm to solve a problem. The *complexity in space* is a function that associates an order of magnitude of the number of memory units used for the operations necessary for the solution of a given problem with the size of an instance of that problem.

There are several sets of hypotheses concerning the "standard configuration" that we use as a basis for measuring the complexity of an algorithm. The most commonly used framework is the one known as "worst-case". Here, the complexity of an algorithm is the number of operations carried out on the instance that represents the worst configuration, amongst those of a fixed size, for its execution; this is the framework used in most of this book. However, it is not the only framework for analyzing the complexity of an algorithm. Another framework often used is "average analysis". This kind of analysis consists of finding, for a fixed size (of the instance) n, the average execution time of an algorithm on all the instances of size n; we assume that for this analysis the probability of each instance occurring follows a specific distribution pattern. More often than not, this distribution pattern is considered to be uniform. There are three main reasons for the worst-case analysis being used more often than the average analysis. The first is psychological: the worst-case result tells us for certain that the algorithm being analyzed can never have a level of complexity higher than that shown by this analysis; in other words, the result we have obtained gives us an upper bound on the complexity of our algorithm. The second reason is mathematical: results from a worst-case analysis are often easier to obtain than those from an average analysis, which very often requires mathematical tools and more complex analysis. The third reason is "analysis portability": the validity of an average analysis is limited by the assumptions made about the distribution pattern of the instances; if the assumptions change, then the original analysis is no longer valid.

1.2. Problem complexity

The definition of the complexity of an algorithm can be easily transposed to problems. Informally, *the complexity of a problem is equal to the complexity of the best algorithm that solves it* (this definition is valid independently of which framework we use).

Let us take a size n and a function $f(n)$. Thus:

– $\mathbf{TIMEf(n)}$ is the class of problems for which the complexity (in time) of an instance of size n is in $O(f(n))$.

1. Orders of magnitude are defined as follows: given two functions f and g: $f = O(g)$ if and only if $\exists (k, k') \in (\mathbb{R}^+, \mathbb{R})$ such that $f \leqslant kg + k'$; $f = o(g)$ if and only if $\lim_{n\to\infty}(f/g) = 0$; $f = \Omega(g)$ if and only if $g = o(f)$; $f = \Theta(g)$ if and only if $f = O(g)$ and $g = O(f)$.

– **SPACEf(n)** is the class of problems that can be solved, for an instance of size n, by using a memory space of $O(f(n))$.

We can now specify the following general classes of complexity:

– **P** is the class of all the problems that can be solved in a time that is a polynomial function of their instance size, that is $\mathbf{P} = \cup_{k=0}^{\infty} \mathbf{TIMEn^k}$.

– **EXPTIME** is the class of problems that can be solved in a time that is an exponential function of their instance size, that is $\mathbf{EXPTIME} = \cup_{k=0}^{\infty} \mathbf{TIME2^{n^k}}$.

– **PSPACE** is the class of all the problems that can be solved using a memory space that is a polynomial function of their instance size, that is $\mathbf{PSPACE} = \cup_{k=0}^{\infty} \mathbf{SPACEn^k}$.

With respect to the classes that we have just defined, we have the following relations: $\mathbf{P} \subseteq \mathbf{PSPACE} \subseteq \mathbf{EXPTIME}$ and $\mathbf{P} \subset \mathbf{EXPTIME}$. Knowing whether the inclusions of the first relation are strict or not is still an open problem.

Almost all combinatorial optimization problems can be classified, from an algorithmic complexity point of view, into two large categories. *Polynomial* problems can be solved optimally by algorithms of polynomial complexity, that is in $O(n^k)$, where k is a constant independent of n (this is the class **P** that we have already defined). *Non-polynomial* problems are those for which the best algorithms (those giving an optimum solution) are of a "super-polynomial" complexity, that is in $O(f(n)^{g(n)})$, where f and g are increasing functions in n and $\lim_{n\to\infty} g(n) = \infty$. All these problems contain the class **EXPTIME**.

The definition of any algorithmic problem (and even more so in the case of any combinatorial optimization problem) comprises two parts. The first gives the instance of the problem, that is the type of its variables (a graph, a set, a logical expression, etc.). The second part gives the type and properties to which the expected solution must conform. In the complexity theory case, algorithmic problems can be classified into three categories:

– *decision problems*;

– *optimum value calculation problems*;

– *optimization problems*.

Decision problems are questions concerning the existence, for a given instance, of a configuration such that this configuration itself, or its value, conforms to certain properties. The solution to a decision problem is an answer to the question associated with the problem. In other words, this solution can be:

– either "*yes*, such a configuration does exist";

– or "*no*, such a configuration does not exist".

Let us consider as an example the conjunctive normal form satisfiability problem, known in the literature as SAT: "Given a set U of n Boolean variables $x_1, \ldots, x_n$ and a set $\mathcal{C}$ of m *clauses*[2] $C_1, \ldots, C_m$, is there a model for the expression $\phi = C_1 \wedge \ldots \wedge C_m$; i.e. is there an assignment of the values 0 or 1 to the variables such that $\phi = 1$?". For an instance ϕ of this problem, if ϕ allows a model then the solution (the correct answer) is *yes*, otherwise the solution is *no*.

Let us now consider the MIN TSP problem, defined as follows: given a complete graph K_n over n vertices for which each edge $e \in E(K_n)$ has a value $d(e) > 0$, we are looking for a Hamiltonian cycle $H \subset E$ (a partial closely related graph such that each vertex is of 2 degrees) that minimizes the quantity $\sum_{e \in H} d(e)$. Let us assume that for this problem we have, as well as the complete graph K_n and the vector $\overline{d}$, costs on the edges K_n of a constant K and that we are looking not to determine the smallest (in terms of total cost) Hamiltonian cycle, but rather to answer the following question: "Does there exist a Hamiltonian cycle of total distance less than or equal to K?". Here, once more, the solution is either *yes* if such a cycle exists, or *no* if it does not.

For *optimum value calculation problems*, we are looking to calculate *the value of the optimum solution* (and not the solution itself).

In the case of the MIN TSP for example, the optimum associated value calculation problem comes down to calculating the cost of the smallest Hamiltonian cycle, and not the cycle itself.

Optimization problems, which are naturally of interest to us in this book, are those for which we are looking to establish the best solution amongst those satisfying certain properties given by the very definition of the problem. An optimization problem may be seen as a mathematical program of the form:

$$\begin{cases} \text{opt} & v(\overline{x}) \\ & \overline{x} \in C_I \end{cases}$$

where $\overline{x}$ is the vector describing the solution[3], $v(\overline{x})$ is the *objective function*, C_I is the problem's constraint set, set out for the instance I (in other words, C_I sets out both the instance and the properties of the solution that we are looking to find for this instance), and $\text{opt} \in \{\max, \min\}$. An *optimum solution* of I is a vector $\overline{x}^* \in \text{argopt}\{v(\overline{x}) : \overline{x} \in C_I\}$. The quantity $v(\overline{x}^*)$ is known as the *objective value* or *value* of the problem. A solution $\overline{x} \in C_I$ is known as a *feasible solution*.

2. We associate two *literals* x and $\bar{x}$ with a Boolean variable x, that is the variable itself and its negation; a clause is a disjunction of the literals.

3. For the combinatorial optimization problems that concern us, we can assume that the components of this vector are 0 or 1 or, if need be, integers.

Let us consider the problem MIN WEIGHTED VERTEX COVER[4]. An instance of this problem (given by the information from the incident matrix A, of dimension $m \times n$, of a graph $G(V, E)$ of order n with $|E| = m$ and a vector $\overline{w}$, of dimension n of the costs of the edges of V), can be expressed in terms of a linear program in integers as follows:

$$\begin{cases} \min & \overline{w} \cdot \overline{x} \\ & A.\overline{x} \geqslant \overline{1} \\ & \overline{x} \in \{0,1\}^n \end{cases}$$

where $\overline{x}$ is a vector from 0, 1 of dimension n such that $x_i = 1$ if the vertex $v_i \in V$ is included in the solution, $x_i = 0$ if it is not included. The block of m constraints $A.\overline{x} \geqslant \overline{1}$ expresses the fact that for each edge at least one of these extremes must be included in the solution. The feasible solutions are all the transversals of G and the optimum solution is a transversal of G of minimum total weight, that is a transversal corresponding to a feasible vector consisting of a maximum number of 1.

The solution to an optimization problem includes an evaluation of the optimum value. Therefore, an optimum value calculation problem can be associated with an optimization problem. Moreover, optimization problems always have a decisional variant as shown in the MIN TSP example above.

1.3. The classes P, NP and NPO

Let us consider a decision problem Π. If for any instance I of Π a solution (that is a correct answer to the question that states Π) of I can be found algorithmically in polynomial time, that is in $O(|I|^k)$ stages, where $|I|$ is the size of I, then Π is called a *polynomial problem* and the algorithm that solves it a *polynomial algorithm* (let us remember that polynomial problems make up the **P** class).

For reasons of simplicity, we will assume in what follows that the solution to a decision problem is:

– either "*yes*, such a solution exists, and this is it";

– or "*no*, such a solution does not exist".

In other words, if, to solve a problem, we could consult an "oracle", it would provide us with an answer of not just a *yes* or *no* but also, in the first case, a certificate

4. Given a graph $G(V, E)$ of order n, in the MIN VERTEX COVER problem we are looking to find a smallest transversal of G, that is a set $V' \subseteq V$ such that for every edge $(u, v) \in E$, either $u \in V'$, or $v \in V'$ of minimum size; we denote by MIN WEIGHTED VERTEX COVER the version of MIN VERTEX COVER where a positive weight is associated with each vertex and the objective is to find a transversal of G that minimizes the sum of the vertex weights.

proving the veracity of the *yes*. This testimony is simply a solution proposal that the oracle "asserts" as being the real solution to our problem.

Let us consider the decision problems for which the validity of the certificate can be verified in polynomial time. These problems form the class **NP**.

DEFINITION 1.1.– *A decision problem* Π *is in* ***NP*** *if the validity of all solutions of* Π *is verifiable in polynomial time.*

For example, the SAT problem belongs to **NP**. Indeed, given the assignment of the values 0, 1 to the variables of an instance ϕ of this problem, we can, with at most nm applications of the connector $\vee$, decide whether the proposed assignment is a model for ϕ, that is whether it satisfies all the clauses.

Therefore, we can easily see that the decisional variant of MIN TSP seen previously also belongs to **NP**.

Definition 1.1 can be extended to optimization problems. Let us consider an optimization problem Π and let us assume that each instance I of Π conforms to the three following properties:

1) The feasibility of a solution can be verified in polynomial time.

2) The value of a feasible solution can be calculated in polynomial time.

3) There is at least one feasible solution that can be calculated in polynomial time.

Thus, Π belongs to the class **NPO**. In other words, *the class* ***NPO*** *is the class of optimization problems for which the decisional variant is in* ***NP***. We can therefore define the class **PO** of optimization problems for which the decisional variant belongs to the class **P**. In other words, **PO** is the class of problems that can be optimally solved in polynomial time.

We note that the class **NP** has been defined (see definition 1.1) without explicit reference to the optimum solution of its problems, but by reference to the verification of a given solution. Evidently, the condition of belonging to **P** being stronger than that of belonging to **NP** (what can be solved can be verified), we have the obvious inclusion of : $\mathbf{P} \subseteq \mathbf{NP}$ (Figure 1.1).

"What is the complexity of the problems in $\mathbf{NP} \setminus \mathbf{P}$?". The best general result on the complexity of the solution of problems from **NP** is as follows [GAR 79].

THEOREM 1.1.– *For any problem* $\Pi \in$ ***NP****, there is a polynomial* p_Π *such that each instance* I *of* Π *can be solved by an algorithm of complexity* $O(2^{p_\Pi(|I|)})$.

In fact, theorem 1.1 merely gives an upper limit on the complexity of problems in **NP**, but no lower limit. The diagram in Figure 1.1 is nothing more than a conjecture,

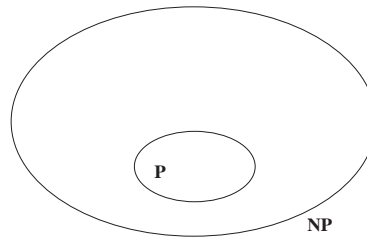

Figure 1.1. ***P*** *and* ***NP*** *(under the assumption* $\mathbf{P} \neq \mathbf{NP}$*)*

and although almost all researchers in complexity are completely convinced of its veracity, it has still not been proved. The question "*Is* ***P*** *equal to or different from* ***NP?***" is the biggest open question in computer science and one of the best known in mathematics.

1.4. Karp and Turing reductions

As we have seen, problems in $\mathbf{NP} \setminus \mathbf{P}$ are considered to be algorithmically more difficult than problems in **P**. A large number of problems in $\mathbf{NP} \setminus \mathbf{P}$ are very strongly bound to each other through the concept of *polynomial reduction*.

The principle of reducing a problem Π to a problem Π' consists of considering the problem Π as a specific case of Π', *modulo* a slight transformation. If this transformation is polynomial, and we know that we can solve Π' in polynomial time, we will also be able to solve Π in polynomial time. Reduction is thus a means of transferring the result of solving one problem to another; in the same way it is a tool for classifying problems according to the level of difficulty of their solution.

We will start with the classic Karp reduction (for the class **NP**) [GAR 79, KAR 72]. This links two decision problems by the possibility of their optimum (and simultaneous) solution in polynomial time. In the following, given a problem Π, let $\mathcal{I}_\Pi$ be all of its instances (we assume that each instance $I \in \mathcal{I}_\Pi$ is identifiable in polynomial time in $|I|$). Let $\mathcal{O}_\Pi$ be the subset of $\mathcal{I}_\Pi$ for which the solution is *yes*; $\mathcal{O}_\Pi$ is also known as the set of *yes*-instances (or positive instances) of Π.

DEFINITION 1.2.– *Given two decision problems* Π_1 *and* Π_2*, a Karp reduction (or polynomial transformation) is a function* $f : \mathcal{I}_{\Pi_1} \rightarrow \mathcal{I}_{\Pi_2}$*, which can be calculated in polynomial time, such that, given a solution for* $f(I)$*, we are able to find a solution for* I *in polynomial time in* $|I|$ *(the size of the instance* I*).*

A Karp reduction of a decision problem Π_1 to a decision problem Π_2 implies the existence of an algorithm $\mathtt{A}_1$ for Π_1 that uses an algorithm $\mathtt{A}_2$ for Π_2. Given any instance $I_1 \in \mathcal{I}_{\Pi_1}$, the algorithm $\mathtt{A}_1$ constructs an instance $I_2 \in \mathcal{I}_{\Pi_2}$; it executes the algorithm $\mathtt{A}_2$, which calculates a solution on I_2, then $\mathtt{A}_1$ transforms this solution into a solution for Π_1 on I_1. If $\mathtt{A}_2$ is polynomial, then $\mathtt{A}_1$ is also polynomial.

Following on from this, we can state another reduction, known in the literature as the *Turing reduction*, which is better adapted to optimization problems. In what follows, we define a problem Π as a couple $(\mathcal{I}_\Pi, \mathrm{Sol}_\Pi)$, where Sol_Π is the set of solutions for Π (we denote by $\mathrm{Sol}_\Pi(I)$ the set of solutions for the instance $I \in \mathcal{I}_\Pi$).

DEFINITION 1.3.– *A Turing reduction of a problem Π_1 to a problem Π_2 is an algorithm $\mathtt{A}_1$ that solves Π_1 by using (possibly several times) an algorithm $\mathtt{A}_2$ for Π_2 in such a way that if $\mathtt{A}_2$ is polynomial, then $\mathtt{A}_1$ is also polynomial.*

The Karp and Turing reductions are transitive: if Π_1 is reduced (by one of these two reductions) to Π_2 and Π_2 is reduced to Π_3, then Π_1 reduces to Π_3. We can therefore see that both reductions preserve membership of the class **P** in the sense that if Π reduces to Π' and $\Pi' \in \mathbf{P}$, then $\Pi \in \mathbf{P}$.

For more details on both the Karp and Turing reductions refer to [AUS 99, GAR 79, PAS 04]. In Garey and Johnson [GAR 79] (Chapter 5) there is also a very interesting historical summary of the development of the ideas and terms that have led to the structure of complexity theory as we know it today.

1.5. NP-completeness

From the definition of the two reductions in the preceding section, if Π' reduces to Π, then Π can reasonably be considered as at least as difficult as Π' (regarding their solution by polynomial algorithms), in the sense that a polynomial algorithm for Π would have sufficed to solve not only Π itself, but equally Π'. Let us confine ourselves to the Karp reduction. By using it, we can highlight a problem $\Pi^* \in \mathbf{NP}$ such that any other problem $\Pi \in \mathbf{NP}$ reduces to Π^* [COO 71, GAR 79, FAG 74]. Such a problem is, as we have mentioned, the most difficult problem of the class **NP**. Therefore we can show [GAR 79, KAR 72] that there are other problems $\Pi^{*\prime} \in \mathbf{NP}$ such that Π^* reduces to $\Pi^{*\prime}$. In this way we expose a family of problems such that any problem in **NP** reduces (remembering that the Karp reduction is transitive) to one of its problems. This family has, of course, the following properties:

– It is made up of the most difficult problems of **NP**.

– A polynomial algorithm for at least one of its problems would have been sufficient to solve, in polynomial time, all the other problems of this family (and indeed any problem in **NP**).

The problems from this family are ***NP**-complete* problems and the class of these problems is called the ***NP**-complete* class.

DEFINITION 1.4.– *A decision problem* Π *is **NP**-complete if, and only if, it fulfills the following two conditions:*

1) $\Pi \in$ ***NP** ;*

2) $\forall \Pi' \in$ ***NP**,* Π' *reduces to* Π *by a Karp reduction.*

Of course, a notion of **NP**-completeness very similar to that of definition 1.4 can also be based on the Turing reduction.

The following application of definition 1.3 is very often used to show the **NP**-completeness of a problem. Let $\Pi_1 = (\mathcal{I}_{\Pi_1}, \mathrm{Sol}_{\Pi_1})$ and $\Pi_2 = (\mathcal{I}_{\Pi_2}, \mathrm{Sol}_{\Pi_2})$ be two problems, and let (f, g) be a pair of functions, which can be calculated in polynomial time, where:

– $f : \mathcal{I}_{\Pi_1} \rightarrow \mathcal{I}_{\Pi_2}$ is such that for any instance $I \in \mathcal{I}_{\Pi_1}$, $f(I) \in \mathcal{I}_{\Pi_2}$;

– $g : \mathcal{I}_{\Pi_1} \times \mathrm{Sol}_{\Pi_2} \rightarrow \mathrm{Sol}_{\Pi_1}$ is such that for every pair $(I, S) \in (\mathcal{I}_{\Pi_1} \times \mathrm{Sol}_{\Pi_2}(f(I)))$, $g(I, S) \in \mathrm{Sol}_{\Pi_1}(I)$.

Let us assume that there is a polynomial algorithm $\mathtt{A}$ for the problem Π_2. In this case, the algorithm $f \circ \mathtt{A} \circ g$ is a (polynomial) Turing reduction.

A problem that fulfills condition 2 of definition 1.4 (without necessarily checking condition 1) is called **NP**-hard[5]. It follows that *a decision problem* Π *is **NP**-complete if and only if it belongs to **NP** and it is **NP**-hard.*

With the class **NP**-complete, we can further refine (Figure 1.2) the world of **NP**. Of course, if $\mathbf{P} = \mathbf{NP}$, the three classes from Figure 1.2 coincide; moreover, under the assumption $\mathbf{P} \neq \mathbf{NP}$, the classes **P** and **NP**-complete do not intersect.

Let us denote by **NP**-intermediate the class $\mathbf{NP} \setminus (\mathbf{P} \cup \mathbf{NP} - complete)$. Informally, this concerns the class of problems of intermediate difficulty, that is problems that are more difficult than those from **P** but easier than those from **NP**-complete. More formally, for two complexity classes **C** and $\mathbf{C}'$ such that $\mathbf{C}' \subseteq \mathbf{C}$, and a reduction R preserving the membership of $\mathbf{C}'$, a problem is **C**-intermediate if it is neither **C**-complete under R, nor belongs to $\mathbf{C}'$. Under the Karp reduction, the class **NP**-intermediate is not empty [LAD 75].

Let us note that the idea of **NP**-completeness goes hand in hand with *decision problems.* When dealing with optimization problems, the appropriate term, used in

5. These are the starred problems in the appendices of [GAR 79].

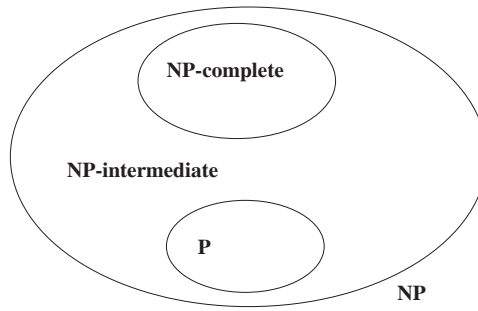

Figure 1.2. ***P****,* ***NP*** *and* ***NP-complete*** *(under the assumption* $\boldsymbol{P \neq NP}$*)*

the literature, is ***NP****-hard*[6]*. A problem of* ***NPO*** *is* ***NP****-hard if and only if its decisional variant is an* ***NP****-complete* problem.

The problem SAT was the first problem shown to be **NP**-complete (the proof of this important result can be found in [COO 71]). The reduction used (often called *generic reduction*) is based on the theory of recursive languages and Turing machines (see [HOP 79, LEW 81] for more details and depth on the Turing machine concept; also, language-problem correspondence is very well described in [GAR 79, LEW 81]). The general idea of generic reduction, also often called the "Cook–Levin technique (or theory)", is as follows: *For a generic decision (language) problem belonging to* ***NP****, we describe, using a normal conjunctive form, the working of a non-deterministic algorithm (Turing machine) that solves (decides) this problem (language).*

The second problem shown to be **NP**-complete [GAR 79, KAR 72] was the variant of SAT, written 3SAT, where no clause contains more than three literals. The reduction here is from SAT [GAR 79, PAP 81]. More generally, for all $k \geqslant 3$, the kSAT problems (that is the problems defined on normal conjunctive forms where each clause contains no more than k literals) are all **NP**-complete.

It must be noted that the problem 2SAT, where all normal conjunctive form clauses contain at most two literals, is polynomial [EVE 76]. It should also be noted that in [KAR 72], where there is a list of the first 21 **NP**-complete problems, the problem of linear programming in real numbers was mentioned as a probable problem from the

6. There is a clash with this term when it is used for optimization problems and when it is used in the sense of property 2 of definition 1.4, where it means that a decision problem Π is harder than any other decision problem $\Pi' \in$ **NP**.

class **NP-intermediate**. It was shown, seven years later ([KHA 79] and also [ASP 80], an English translation of [KHA 79]), that this problem is in **P**.

The reference on **NP**-completeness is the volume by Garey and Johnson [GAR 79]. In the appendix, *A list of **NP**-complete problems*, there is a long list of **NP**-complete problems with several commentaries for each one and for their limited versions. For many years, this list has been regularly updated by Johnson in the *Journal of Algorithms* review. This update, supplemented by numerous commentaries, appears under the title: *The NP-completeness column: an ongoing guide*.

"What is the relationship between optimization and decision for **NP**-complete problems?". The following theory [AUS 95, CRE 93, PAZ 81] attempts to give an answer.

THEOREM 1.2.– *Let Π be a problem of **NPO** and let us assume that the decisional version of Π, written Π_d, is **NP**-complete. It follows that a polynomial Turing reduction exists between Π_d and Π.*

In other words, the decision versions (such as those we have considered in this chapter) and optimization versions of an **NP**-complete problem are of equivalent algorithmic difficulty. However, the question of the existence of a problem **NPO** for which the optimization version is more difficult to solve than its decisional counterpart remains open.

1.6. Two examples of NP-complete problems

Given a problem Π, the most conventional way to show its **NP**-completeness consists of making a Turing or Karp reduction of an **NP**-complete Π' problem to Π. In practical terms, the proof of **NP**-completeness for Π is divided into three stages:

1) proof of membership of Π to **NP**;

2) choice of Π';

3) building the functions f and g (see definition 1.3) and showing that they can both be calculated in polynomial time.

In the following, we show that MIN VERTEX COVER$(G(V,E),K)$ and MAX STABLE$(G(V,E),K)$, the decisional variants of MIN VERTEX COVER and of MAX STABLE[7], respectively, are **NP**-complete. These two problems are defined as follows. MIN VERTEX COVER$(G(V,E),K)$: given a graph G and a constant $K \leqslant |V|$, does there exist in G a transversal $V' \subseteq V$ less than or equal in size to K? MAX

7. Given a graph $G(V,E)$ of magnitude n, we are trying to find a stable set of maximum size, that is a set $V' \subseteq V$ such that $\forall (u,v) \in V' \times V'$, $(u,v) \notin E$ of maximum size.

STABLE SET$(G(V,E),K)$: given a graph G and a constant $K \leqslant |V|$, does there exist in G a stable set $V' \subseteq V$ of greater than or equal in size to K?

1.6.1. MIN VERTEX COVER

The proof of membership of MIN VERTEX COVER$(G(V,E),K)$ to **NP** is very simple and so has been omitted here. We will therefore show the completeness of this problem for **NP**. We will transform an instance $\phi(U,\mathcal{C})$ from 3SAT, with $U = \{x_1,\ldots,x_n\}$ and $\mathcal{C} = \{C_1,\ldots,C_m\}$, into an instance $(G(V,E),K)$ of MIN VERTEX COVER.

This graph is made up of two component sets, joined by edges. The first component is made up of $2n$ vertices $x_1,\bar{x}_1,\ldots,x_n,\bar{x}_n$ and n edges $(x_i,\bar{x}_i)$, $i = 1,\ldots,n$, which join the vertices in pairs. The second is made up of m vertex-disjoint triangles (that is of m cliques with three vertices). For a clause C_i, we denote the three vertices of the corresponding triangle by c_{i1}, c_{i2} and c_{i3}. In fact, the first set of components, for which each vertex corresponds to a literal, serves to define the truth values of the solution for 3SAT; the second set of components corresponds to the clauses, and each vertex is associated with a literal of its clause. These triangles are used to verify the satisfaction of the clauses. To finish, we add $3m$ "unifying" edges that link each vertex of each "triangle-clause" to its corresponding "literal-vertex". Let us note that exactly three unifying edges go from (the vertices of) each triangle, one per vertex of the triangle. Finally, we state $K = n + 2m$. It is easy to see that the transformation of $\phi(U,\mathcal{C})$ to $G(V,E)$ happens in polynomial time in $\max\{m,n\}$ since $|V| = 2n + 3m$ and $|E| = n + 6m$.

As an example of the transformation described above, let us consider the instance $\phi = (x_1 \vee \bar{x}_2 \vee x_3) \wedge (\bar{x}_1 \vee x_2 \vee \bar{x}_3) \wedge (x_2 \vee x_3 \vee \bar{x}_4) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee x_4)$ from 3SAT. The graph $G(V,E)$ for MIN VERTEX COVER is given in Figure 1.3. In this case, $K = 12$.

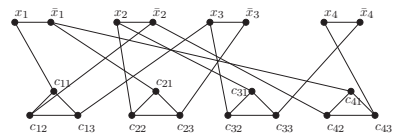

Figure 1.3. *The graph associated with the expression* $(x_1 \vee \bar{x}_2 \vee x_3) \wedge (\bar{x}_1 \vee x_2 \vee \bar{x}_3) \wedge (x_2 \vee x_3 \vee \bar{x}_4) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee x_4)$

We will now show that *G allows a transversal less than or equal in size to $n + 2m$ if and only if the expression ϕ can be satisfied.*

Let us first show that the condition is necessary. Let us assume that there is a polynomial algorithm A that answers the question "Is there in G a transversal $V' \subseteq V$ of size $|V'| \leqslant K$?", and, if so, returns V'. Let us execute it with $K = n + 2m$. If the answer from A is *yes*, then the transversal must be of a size *equal* to $n+2m$. In fact, any transversal needs at least n vertices in order to cover the n edges corresponding to the variables of ϕ (one vertex per edge) and $2m$ vertices to cover the edges of m triangles (two vertices per triangle). As a result, if A answers *yes*, it will have calculated a transversal of exactly $n + 2m$ vertices.

In the light of the previous observation, given such a transversal V', we state that $x_i = 1$ if the extremity x_i of the edge $(x_i, \bar{x}_i)$ is taken in V'; if the extremity $\bar{x}_i$ is included, then we state that $\bar{x}_i = 1$, that is $x_i = 0$. We claim that this assignment of the truth values to the variables satisfies ϕ. Indeed, since only one extremity of each edge $(x_i, \bar{x}_i)$ is taken in V', *only one literal is set to 1 for each variable* and, in consequence, the assignment in question is coherent (one, and only one, truth value is assigned to each literal). Moreover, let us consider a triangle T_i of G corresponding to the clause C_i; let us denote its vertices by c_{i1}, c_{i2} and c_{i3}, and let us assume that the last two belong to V'. Let us also assume that the unifying edge having as an extremity the vertex c_{i1} is the edge (c_{i1}, ℓ_k), ℓ_k being one of the literals associated with the variable x_k. Since $c_{i1} \notin V'$, ℓ_k belongs to it, that is $\ell_k = 1$, and the existence of the edge (c_{i1}, ℓ_k) means that the literal ℓ_k belongs to the clause C_i. This is proved by setting ℓ_k to 1. By iterating this argument for each clause, the need for the condition is proved. Furthermore, let us note that obtaining the assignment of the truth values to the variables of ϕ is done in polynomial time.

Let us now show that the condition is also good enough. Given an assignment of truth values satisfying the expression ϕ, let us construct in G a transversal V' of size $n + 2m$. To start with, for each variable x_i, if $x_i = 1$, then the extremity x_i of the edge $(x_i, \bar{x}_i)$ is put in V'; otherwise, the extremity $\bar{x}_i$ of the edge $(x_i, \bar{x}_i)$ is put there. We thereby cover the edges of type $(x_i, \bar{x}_i)$, $i = 1, \ldots, n$, and one unifying edge per triangle. Let T_i (corresponding to the clause C_i), $i = 1, \ldots, m$, be a triangle and let (ℓ_k, c_{i1}) be the unifying edge covered by the setting to 1 of ℓ_k. We therefore put the vertices c_{i2} and c_{i3} in V'; these vertices cover both the edges of T_i and the two unifying edges having as extremities c_{i2} and c_{i3}, respectively. By iterating this operation for each triangle, a transversal V' of size $n + 2m$ is eventually constructed in polynomial time.

1.6.2. MAX STABLE

The proof of membership of MAX STABLE$(G(V, E), K)$ is so simple that it is omitted here.

Let us consider a graph $G(V, E)$ of magnitude n having m edges and let us denote by A its incidence matrix[8]. Let us also consider the expression of MIN VERTEX COVER as a linear program in whole numbers and the transformations that follow:

$$\Leftrightarrow \left\{\begin{array}{ll}\min & \overline{1}\cdot\overline{y}\\ & A\cdot\overline{y}\geqslant\overline{1}\\ & \overline{y}\in\{0,1\}^n\end{array}\right. \Leftrightarrow \left\{\begin{array}{ll}\min & \overline{1}\cdot\overline{y}\\ & 2.\overline{1}-A\cdot\overline{y}\leqslant\overline{1}\\ & \overline{y}\in\{0,1\}^n\end{array}\right.$$

$$\Leftrightarrow \left\{\begin{array}{ll}\min & \overline{1}\cdot\overline{y}\\ & A\cdot\left(\overline{1}-\overline{y}\right)\leqslant\overline{1}\\ & \overline{y}\in\{0,1\}^n\end{array}\right. \overset{\overline{y}=\overline{1}-\overline{x}}{\Longleftrightarrow} \left\{\begin{array}{ll}\min & \overline{1}\cdot\left(\overline{1}-\overline{x}\right)\\ & A\cdot\overline{x}\leqslant\overline{1}\\ & \overline{x}\in\{0,1\}^n\end{array}\right.$$

$$\Leftrightarrow \left\{\begin{array}{ll}\max & \overline{1}\cdot\overline{x}\\ & A\cdot\overline{x}\leqslant\overline{1}\\ & \overline{x}\in\{0,1\}^n\end{array}\right.$$

We note that the last program in the series is nothing more than the linear program (in whole numbers) of MAX STABLE. Furthermore, this series of equivalents indicates that if a solution vector $\overline{x}$ for MAX STABLE is given, then the vector $\overline{y} = \overline{1} - \overline{x}$ (that is the vector $\overline{y}$ where we interchange the "1" and the "0" regarding $\overline{x}$) is a feasible solution for MIN VERTEX COVER. Furthermore, if $\overline{x}$ contains at least K "1" (that is the size of the stable set is at least equal to K), then the solution vector deduced for MIN VERTEX COVER contains at most $n - K$ "1" (that is the size of the transversal is at most equal to $n - K$). Since the function $\overline{x} \mapsto \overline{1} - \overline{x}$ is polynomial, then so is the described transformation.

1.7. A few words on strong and weak NP-completeness

Let Π be a problem and I an instance of Π of size $|I|$. We denote by $\max(I)$ the highest number that appears in I. Let us note that $\max(I)$ can be exponential in n. An algorithm for Π is known as *pseudo-polynomial* if it is polynomial in $|I|$ and $\max(I)$ (if $\max(I)$ is exponential in $|I|$, then this algorithm is exponential for I).

DEFINITION 1.5.– *An optimization problem is **NP**-complete in the strong sense (strongly **NP**-complete) if the problem is **NP**-complete because of its structure and not because of the size of the numbers that appear in its instances. A problem is **NP**-complete in the weak sense (weakly **NP**-complete) if it is **NP**-complete because of its valuations (that is* $\max(I)$ *affects the complexity of the algorithms that solve it).*

Let us consider on the one hand the SAT problems, or MIN VERTEX COVER problems, or even the MAX STABLE problems seen previously, and on the other hand the KNAPSACK problem for which the decisional variant is defined as: "given a maximum cost b, n objects $\{1, \ldots, n\}$ of respective values a_i and respective costs $c_i \leqslant b$,

8. This matrix is of dimensions $m \times n$.

$i = 1, \ldots, n$, and a constant K, is there a subset of objects for which the sum of the values is at least equal to K without the sum of the costs of these objects exceeding b?". This problem is the most well-known weakly **NP**-complete problem. Its intrinsic difficulty is not due to its structure, as is the case for the previous three problems where no large number affects the description of their instances, but rather due to the values of a_i and c_i that do affect the specification of the instance of KNAPSACK.

In Chapter 4 (see also Volume 2, Chapter 8), we find a dynamic programming algorithm that solves this problem in linear time for the highest value of a_i and in logarithmic time for the highest value of c_i. This means that if this value is a polynomial of n, then the algorithm is also polynomial, and if the value is exponential in n, the algorithm itself is of exponential complexity.

The result below [GAR 79, PAS 04] follows the borders between strongly and weakly **NP**-complete problems. *If a problem* Π *is strongly* ***NP****-complete, then it cannot be solved by a pseudo-polynomial algorithm, unless* $\mathbf{P} = \mathbf{NP}$.

1.8. A few other well-known complexity classes

In this section, we briefly present a few supplementary complexity classes that we will encounter in the following chapters (for more details, see [BAL 88, GAR 79, PAP 94]). Introductions to some complexity classes can also be found in [AUS 99, VAZ 01].

Let us consider a decision problem Π and a generic instance I of Π defined on a data structure S (a graph, for example) and a decision constant K. From the definition of the class **NP**, we can deduce that if there is a solution giving the answer *yes* for Π on I, and if this solution is submitted for verification, then the answer for any correct verification algorithm will always be *yes*. On the other hand, if such a solution does not exist, then any solution proposal submitted for verification will always bring about a *no* answer from any correct verification algorithm.

Let us consider the following decisional variant of MIN TSP, denoted by co-MIN TSP: "given K_n, $\overline{d}$ and K, is it true that there is no Hamiltonian cycle of a total distance less than or equal to K?". How can we guarantee that the answer for an instance of this problem is *yes*? This questions leads on to that of this problem's membership of the class **NP**. We come across the same situation for a great many problems in $\mathbf{NP} \setminus \mathbf{P}$ (assuming that $\mathbf{P} \neq \mathbf{NP}$).

We denote by $\mathcal{I}_\Pi$ the set of all the instances of a decision problem $\Pi \in \mathbf{NP}$ and by $\mathcal{O}_\Pi$ the subset of $\mathcal{I}_\Pi$ for which the solution is *yes*, that is the set of *yes*-instances (or positive instances) of Π. We denote by $\bar{\Pi}$ the problem having as *yes*-instances the set $\mathcal{O}_{\bar{\Pi}} = \mathcal{I}_\Pi \setminus \mathcal{O}_\Pi$, that is the set of *no*-instances of Π. All these problems make

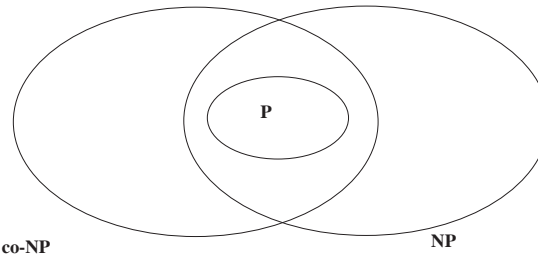

Figure 1.4. ***P, NP*** *and* ***co-NP*** *(under the conditions* $\mathbf{P} \neq \mathbf{NP}$ *and* $\mathbf{NP} \neq$ ***co-NP****)*

up the class **co-NP**. In other words, $\textbf{co-NP} = \{\bar{\Pi} : \Pi \in \textbf{NP}\}$. It is surmised that $\textbf{NP} \neq \textbf{co-NP}$. This surmise is considered as being "stronger" than $\textbf{P} \neq \textbf{NP}$, in the sense that it is possible that $\textbf{P} \neq \textbf{NP}$, even if $\textbf{NP} = \textbf{co-NP}$ (but if $\textbf{P} = \textbf{NP}$, then $\textbf{NP} = \textbf{co-NP}$).

Obviously, for a decision problem $\Pi \in \textbf{P}$, the problem $\bar{\Pi}$ also belongs to **P**; as a result, $\textbf{P} \subseteq \textbf{NP} \cap \textbf{co-NP}$.

A decision problem Π belongs to **RP** if there is a polynomial p and an algorithm A such that, for any instance I: if $I \in \mathcal{O}_\Pi$, then the algorithm gives a decision in polynomial time for at least half of the candidate solutions (certificates) S, such that $|S| \leqslant p(|I|)$, that are submitted to it for verification of their feasibility; if, on the other hand, $I \notin \mathcal{O}_\Pi$ (that is if $I \in \mathcal{I}_\Pi \setminus \mathcal{O}_\Pi$), then for any proposed solution S with $|S| \leqslant p(|I|)$, A rejects S in polynomial time. A problem $\Pi \in$ **co-RP** if and only if $\bar{\Pi} \in$ **RP**, where $\bar{\Pi}$ is defined as before. We have the following relationship between **P**, **RP** and **NP**: $\textbf{P} \subseteq \textbf{RP} \subseteq \textbf{NP}$. For a very simple and intuitive proof of this relationship, see [VAZ 01].

The class **ZPP** (an abbreviation of *zero-error probabilistic polynomial time*) is the class of decision problems that allow a randomized algorithm (for this subject see [MOT 95] and Chapter 2) that always ends up giving the correct answer to the question "$I \in \mathcal{O}_\Pi$?", with, on average, a polynomial complexity. A problem Π belongs to **ZPP** if and only if Π belongs to $\textbf{RP} \cap \textbf{co-RP}$. In other words, $\textbf{ZPP} = \textbf{RP} \cap \textbf{co-RP}$.

1.9. Bibliography

[ASP 80] ASPVALL B., STONE R.E., "Khachiyan's linear programming algorithm", *J. Algorithms*, vol. 1, p. 1–13, 1980.

[AUS 95] AUSIELLO G., CRESCENZI P., PROTASI M., "Approximate solutions of NP optimization problems", *Theoret. Comput. Sci.*, vol. 150, p. 1–55, 1995.

[AUS 99] AUSIELLO G., CRESCENZI P., GAMBOSI G., KANN V., MARCHETTI-SPACCAMELA A., PROTASI M., *Complexity and Approximation. Combinatorial Optimization Problems and their Approximability properties*, Springer-Verlag, Berlin, 1999.

[BAL 88] BALCÀZAR J.L., DIAZ J., GABARRÒ J., *Structural Complexity*, vol. 1 and 2, Springer-Verlag, Berlin, 1988.

[COO 71] COOK S.A., "The complexity of theorem-proving procedures", *Proc. STOC'71*, p. 151–158, 1971.

[CRE 93] CRESCENZI P., SILVESTRI R., "Average measure, descriptive complexity and approximation of minimization problems", *Int. J. Foundations Comput. Sci.*, vol. 4, p. 15–30, 1993.

[EVE 76] EVEN S., ITAI A., SHAMIR A., "On the complexity of timetable and multicommodity flow problems", *SIAM J. Comput.*, vol. 5, p. 691–703, 1976.

[FAG 74] FAGIN R., "Generalized first-order spectra and polynomial-time recognizable sets", KARP R.M., Ed., *Complexity of Computations*, p. 43–73, American Mathematics Society, 1974.

[GAR 79] GAREY M.R., JOHNSON D.S., *Computers and Intractability. A Guide to the Theory of NP-completeness*, W. H. Freeman, San Francisco, 1979.

[HOP 79] HOPCROFT J.E., ULLMAN J.D., *Introduction to Automata Theory, Languages and Computation*, Addison-Wesley, 1979.

[KAR 72] KARP R.M., "Reducibility among combinatorial problems", MILLER R.E., THATCHER J.W., Eds., *Complexity of computer computations*, p. 85–103, Plenum Press, New York, 1972.

[KHA 79] KHACHIAN L.G., "A polynomial algorithm for linear programming", *Dokladi Akademiy Nauk SSSR*, vol. 244, p. 1093–1096, 1979.

[LAD 75] LADNER R.E., "On the structure of polynomial time reducibility", *J. Assoc. Comput. Mach.*, vol. 22, p. 155–171, 1975.

[LEW 81] LEWIS H.R., PAPADIMITRIOU C.H., *Elements of the Theory of Computation*, Prentice-Hall, New Jersey, 1981.

[MOT 95] MOTWANI R., RAGHAVAN P., *Randomized Algorithms*, Cambridge University Press, Cambridge, 1995.

[PAP 81] PAPADIMITRIOU C.H., STEIGLITZ K., *Combinatorial Optimization: Algorithms and Complexity*, Prentice-Hall, New Jersey, 1981.

[PAP 94] PAPADIMITRIOU C.H., *Computational Complexity*, Addison-Wesley, 1994.

[PAS 04] PASCHOS V.TH., *Complexité et approximation polynomiale*, Hermes, Paris, 2004.

[PAZ 81] PAZ A., MORAN S., "Non deterministic polynomial optimization problems and their approximations", *Theoret. Comput. Sci.*, vol. 15, p. 251–277, 1981.

[VAZ 01] VAZIRANI V., *Approximation Algorithms*, Springer-Verlag, Berlin, 2001.