Combinatorial Optimization
volume 1

Concepts of Combinatorial Optimization

Edited by
Vangelis Th. Paschos

ISTE

WILEY

First published 2010 in Great Britain and the United States by ISTE Ltd and John Wiley & Sons, Inc. Adapted and updated from *Optimisation combinatoire* volumes 1 to 5 published 2005-2007 in France by Hermes Science/Lavoisier © LAVOISIER 2005, 2006, 2007

ISTE Ltd
27-37 St George's Road
London SW19 4EU
UK

www.iste.co.uk

John Wiley & Sons, Inc.
111 River Street
Hoboken, NJ 07030
USA

www.wiley.com

Library of Congress Cataloging-in-Publication Data

Combinatorial optimization / edited by Vangelis Th. Paschos.
v. cm.
Includes bibliographical references and index.
Contents: v. 1. Concepts of combinatorial optimization
ISBN 978-1-84821-146-9 (set of 3 vols.) -- ISBN 978-1-84821-147-6 (v. 1)
1. Combinatorial optimization. 2. Programming (Mathematics)
I. Paschos, Vangelis Th.
QA402.5.C545123 2010
519.6'4--dc22

2010018423

British Library Cataloguing-in-Publication Data
A CIP record for this book is available from the British Library
ISBN 978-1-84821-146-9 (Set of 3 volumes)
ISBN 978-1-84821-147-6 (Volume 1)

Printed and bound in Great Britain by CPI Antony Rowe, Chippenham and Eastbourne.

Table of Contents

Chapter 2

Randomized Complexity

A deterministic algorithm is "a method of solution, by the application of a rule, a finite number of times" (Chapter 1). This definition extends to probabilistic algorithms by allowing probabilistic rules, or by giving a distribution of probability over a set of deterministic algorithms. By definition, probabilistic algorithms are potentially more *powerful* than their deterministic counterparts: the class of probabilistic algorithms contains the class of deterministic algorithms. It is therefore natural to consider probabilistic algorithms for solving the hardest combinatorial optimization problems, all the more so because probabilistic algorithms are often simpler than their deterministic counterparts.

A deterministic algorithm is correct if it solves each instance in a valid way. In the context of probabilistic algorithms for which the execution depends both on the instance and on the randomization of the algorithm, we consider the correct complexity of algorithms for any instance and any randomization, but also the complexity of algorithms such that for each instance I, the probability that the algorithm correctly solves I is higher than a constant (typically $1/2$).

In the context of combinatorial optimization, we consider algorithms for which the result *approximates* the solution of the problem set. We then examine both the complexity of the algorithm and the *quality* of the approximations that it gives. We can, without loss of generality, limit ourselves to the problems of minimizing a cost function, in which case we look to minimize both the complexity of the algorithm and the cost of the approximation that it produces. Depending on whether we examine the complexity or the cost of the approximation generated, the terms are different but the

Chapter written by Jérémy BARBAY.

techniques are similar: in both cases, we are looking to minimize a measure of the performance of the algorithm. This performance, for a probabilistic algorithm and a given instance I, is defined as the average of the performance of the corresponding deterministic algorithm on I.

The aim of this chapter is to show that the analysis techniques used to study the complexity of probabilistic algorithms can be just as easily used to analyze the approximation quality of combinatorial optimization algorithms. In section 2.1, we give a more formal definition of the concepts and notations generally used in the study of the complexity of probabilistic algorithms, and we introduce a basic problem that is used to illustrate the most simple results of this chapter. In section 2.2, we give and prove the basic results that allow us to prove lower bounds for the performance of probabilistic algorithms for a given problem. These results can often be used as they stand, but it is important to understand their causes in order to adapt them to less appropriate situations. The most common technique for proving an upper bound to the performance of the best probabilistic algorithm for a given problem is to analyze the performance of the probabilistic algorithm, and for this purpose there are as many techniques as there are algorithms: for this reason we do not describe a general method, but instead give an example of analysis in section 2.3.

2.1. Deterministic and probabilistic algorithms

An *algorithm* is a method for solving a problem using a finite number of rule applications. In the computer science context, an algorithm is a precise description of the stages to be run through in order to carry out a calculation or a specific task. A *deterministic algorithm* is an algorithm such that the choice of each rule to be applied is deterministic. A *probabilistic algorithm* can be defined in a similar way by a probabilistic distribution over the deterministic algorithms, or by giving a probabilistic distribution over the rules of the algorithm. In both cases, the data received by the algorithm make up the *instance* of the problem to be solved, and the choice of algorithm or rules makes up the *randomization* of the execution.

Among deterministic algorithms, generally only the algorithms that *always* give a correct answer are considered, whereas for probabilistic algorithms, algorithms that may be incorrect are also considered, with some constraints. In the context of decision problems, where the answer to each instance is Boolean (accepted or refused), we consider "Monte Carlo" *probabilistic algorithms* [PAP 94, section 11.2] such that for a fixed positive instance, the probability that the algorithm accepts the instance is at least $1/2$, and for a fixed negative instance, the algorithm always refuses the instance (whatever its randomization). This value of $1/2$ is arbitrary: any value ε that is strictly positive is sufficient, since it is enough to repeat the algorithm k times independently to reduce the probability that the algorithm accepts a negative instance to $(1-\varepsilon)^k$.

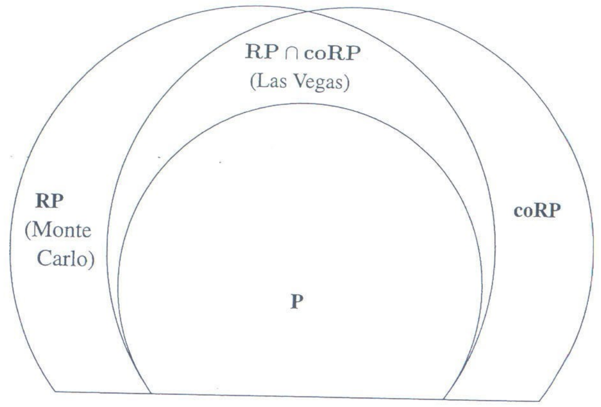

Figure 2.1. *Complexity classes relative to probabilistic algorithms that run in polynomial time*

Problems that can be solved using Monte Carlo algorithms running in polynomial time make up the complexity class **RP**. Among these, problems that can be resolved by deterministic algorithms running in polynomial time make up the complexity class **P**. In a similar way, the class **co-RP** is made up of the set of problems that can be solved in polynomial time by a probabilistic algorithm that always accepts a positive instance, but refuses a negative instance with a probability of at least $1/2$. If a problem is in **ZPP** = **RP**∩**co-RP**, it allows an algorithm of each kind, and so allows an algorithm that is a combination of both, which always accepts positive instances and refuses negative instances, but in an unlimited time. The execution time of algorithms of this type can be random, but they always find a correct result. These algorithms are called "Las Vegas": their result is certain, but their execution time is random, a bit like a martingale with a sufficiently large initial stake.

Another useful complexity class concerns probabilistic algorithms that can make mistakes both by accepting and by refusing instances of the class **BPP**, formed by problems that can be solved in polynomial time by a probabilistic algorithm refusing a positive instance or accepting a negative instance with a probability of at most $1/4$. Just as for **RP**, the value of $1/4$ is arbitrary and can be replaced by $1/2 - p(n)$ for any polynomial $p(n)$ without losing the important properties of **RP** (but the value of $1/2$ would not be suitable here, see [MOT 95, p. 22]).

Despite the fact that these complexity classes are generally defined in terms of decision problems, they can equally well be used to order complexities of a larger class of problems, such as research problems, or combinatorial problems [MOT 95, p. 23].

EXAMPLE.– A classic problem is the *bin-packing* problem: given a list of objects of heights $L = \{x_\bullet, \dots, x_n\}$, between 0 and 1, we must make vertical stacks of objects in such a way that the height of each stack does not exceed 1, and such that there is a minimum number of stacks. This problem is **NP**-complete and the approximation of the optimal number μ of stacks is a classic combinatorial optimization problem. Even if it is not a decision problem, it can be categorized in the same classes. It is enough to consider the following variant: given a list of the weights of objects $L = \{x_\bullet, \dots, x_n\}$, and a number of stacks M, can these n objects be organized into M stacks, without any of them exceeding one unit of height? Any algorithm approximating μ by a value m such that $\Pr[m = \mu] \geqslant 3/4$ (if need be by iterating the same algorithm several times) allows us to decide whether M stacks will suffice ($m \leqslant M$) without ever being wrong when M is not sufficient ($M < \mu \leqslant m$), and with a probability of being wrong of at most $1/4$ if M is sufficient ($\Pr[\mu < m = M] \leqslant 1/4$): the decision problem is in **RP**!

2.1.1. *Complexity of a Las Vegas algorithm*

Given a cost function over the operations carried out by the algorithm, the complexity of an algorithm A on an instance I is the sum of the costs of the instructions corresponding to the execution of A on I. The algorithms solving this same problem are compared by their complexity. The *time complexity* $C(A, I)$ of an algorithm A on an instance I corresponds to the number of instructions carried out by A when it is executed to solve I.

EXAMPLE.–

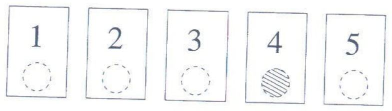

Figure 2.2. *An instance of the hidden coin problem: one silver coin amongst four copper coins, the coins being hidden by cards*

The hidden coin problem is another abstract problem that we will use to illustrate the different ideas of this chapter. We have a row of n cards. Each card hides a coin, which can be a copper coin or a silver coin. The *hidden coin problem* is to decide whether the row contains at least one silver coin.

In this particular case, an algorithm must indicate which coins to uncover, and in which order, depending on which type of coin is revealed. For such a simple problem, we can limit the study to the algorithms that stop uncovering coins as soon as they have discovered a silver coin: any coin uncovered after this would be superfluous. Each of these algorithms is defined by the order σ in which it uncovers the coins as long as there is no silver coin:

– A deterministic algorithm uncovers coins in a fixed order.

– A potential algorithm chooses half of the coins randomly and uniformly, uncovers them, accepts the instance if there is a silver coin, and otherwise refuses. This algorithm always refuses an instance that does not contain a silver coin (therefore a negative instance), and accepts any instance containing at least one silver coin (therefore positive) with a probability of at least $1/2$: therefore it is a Monte Carlo algorithm.

– Another potential algorithm uncovers coins in a random order until it has found a silver coin, or it has uncovered all of the coins: this algorithm always gives the right answer, but the number of coins uncovered is a random variable that depends on the chosen order: it is therefore a Las Vegas algorithm.

In a configuration that has only one silver coin hidden under the second card from the right, the algorithm that uncovers coins from left to right will uncover $n-1$ coins, the algorithm that uncovers coins from right to left will only uncover 2 coins.

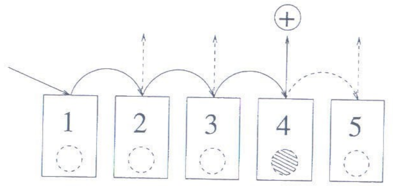

Figure 2.3. *The algorithm choosing coins from left to right: dotted lines show the possible executions, and solid lines show the executions on this instance. The answer of the algorithm is positive because the row contains a silver coin*

Let $F = \{I_\bullet, \dots, I_{|F|}\}$ be a finite set of instances, and A an algorithm for these instances. The complexity $C(A, F)$ of A on F can be defined in several ways. The set of values taken by the complexity of A on the instances of F is $\{C(A, I_\bullet), \dots, C(A, I_{|F|})\}$:

– their maximum $C(A, F) = \max_{I \in F} C(A, I)$ corresponds to the *worst-case complexity*;

– and the average $C(A, F) = \sum_{I \in F} C(A, I) p(I)$ corresponds to the *average complexity* according to a probability distribution $p(I)$ on the instances.

EXAMPLE.– The worst-case complexity of the algorithm choosing the coins from left to right is n.

The complexity of an algorithm on an infinite number of instances cannot be defined in the same way: the number of values of complexity to be considered is potentially infinite. To define this complexity, the set of instances is partitioned into subsets of a finite cardinality indexed by $\mathbb{N}$, for example the instances that can be coded in n machine words, for any integer n. For any integer n, the complexity $f(n)$ (in the worst case, or on average) of the algorithm on the subset of index n is therefore well defined. The complexity $C(A)$ of the algorithm A on the problem is defined as the function f which for each integer n connects $f(n)$.

EXAMPLE.– The set of configurations is finite. For the hidden coin problem with n coins, where n is fixed, algorithm 2.1 will uncover all n coins in the worst case: its complexity in the worst case is therefore $f(n) = n$. Its average complexity on the uniform distribution of the instances containing only one silver coin is $(n+1)/2$.

Algorithm 2.1 Listing of the algorithm that uncovers coins from left to right

```
while there are still coins to uncover, and no silver coin has been found do
    uncover the coin the furthest to the left that has not been uncovered already;
end while
if a silver coin has been uncovered then
    answer positively
else
    answer negatively.
end if
```

2.1.2. *Probabilistic complexity of a problem*

To prove a lower bound to the complexity of a problem, we must be able to consider all the possible algorithms. To this end, the algorithms are represented in the form of trees, where each node is an instruction, each branch is an execution, and each leaf is a result: this is the *decision tree* model.

DEFINITION 2.1.– A questionnaire *is a tree whose leaves are labeled by classes and whose internal nodes are labeled by tests. If the test has k possible results, the internal node has k threads, and the k arcs linking the node to its threads are labeled by these results. The questionnaire allows us to decide to which class a given instance belongs. The instance is subjected to a series of tests, starting with the test contained in the root node. Following the result of the test, the series of tests continues in the corresponding sub-branch. If the sub-branch is a leaf, the label of this leaf is the class associated with the instance.*

DEFINITION 2.2.– A decision tree *is a questionnaire where each internal node corresponds to a deterministic operation that can be executed on any instance in a finite time.*

All deterministic algorithms ending in a finite time can be expressed as a decision tree. Each leaf then corresponds to a possible result of the algorithm. The number of operations carried out by the algorithm on this instance is thus the length of the corresponding branch.

DEFINITION 2.3.– A comparison tree *is a decision tree for which the tests are comparisons between internal elements of the instance.*

Knuth [KNU 73] uses comparison trees to obtain a lower bound on the complexity of comparison-based sorting algorithms. His analysis excludes algorithms such as the sorting by counting algorithm, which directly access the values being sorted, and hence are not in the comparison model.

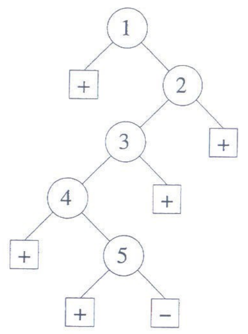

Figure 2.4. *A decision tree for the hidden coin problem when there are five coins*

EXAMPLE.– Any decision tree corresponding to an algorithm for the hidden coin problem must contain at least $n+1$ leaves: one for each possible position for the first silver coin uncovered, and one for the configuration not containing any silver coins. Each test allows us to eliminate exactly one potential position for the silver coin. In this particular case, any decision tree corresponds to a chain. Its height is equal to n. The worst-case complexity of any algorithm for this problem is therefore $\Omega(n)$.

The decision tree model only concerns deterministic algorithms. It can be extended to probabilistic algorithms by distribution over deterministic decision trees.

DEFINITION 2.4.– *For a fixed problem, a* probabilistic algorithm *is defined by a distribution over deterministic algorithms. In the same way, a* probabilistic decision tree *is a distribution over decision trees.*

The complexity of a probabilistic algorithm R on an instance I is the average of the complexities of the deterministic algorithms A on I according to the distribution associated with R; $C(R, I) = \sum_A \Pr\{A\}C(A, I)$.

The model of probabilistic algorithms is more general than that of deterministic algorithms, and allows better performances.

EXAMPLE.– If an instance of the hidden coins problem contains $n/2$ silver coins and $n/2$ copper coins, for each deterministic algorithm there is a configuration of the coins such that it uncovers $n/2$ coins before uncovering a silver coin. The probabilistic algorithm choosing an order σ of positions at random will possibly also uncover up to $n/2$ coins, but with a very low probability of $1/2^{n/2}$. For the worst instance containing $n/2$ silver coins and $n/2$ copper coins, the probabilistic algorithm will uncover less than two coins on average: a lot less than a deterministic algorithm.

DEFINITION 2.5.– *The probabilistic complexity of a problem is equal to the average complexity of the best deterministic algorithm on the worst distribution.*

The complexity of any probabilistic algorithm for a given problem gives an upper bound to the probabilistic complexity of this problem: an example is given in section 2.3.1. Moreover, the minimax theorem allows us to obtain a lower bound to the probabilistic complexity of a given problem: this is presented and proved in section 2.2.

2.2. Lower bound technique

The minimax theorem is a fundamental theoretical tool in the study of probabilistic complexity. It is presented in the context of game theory, in particular for games for two players with a sum total of zero (games where the sum total of the winnings of the two players is always equal to zero). The Yao principle applies this theorem in the context of probabilistic algorithms; the first player applies the algorithm and the second creates an instance in a way that maximizes the complexity of the algorithm: this is related to the min–max algorithms (see Volume 2, Chapter 4).

2.2.1. *Definitions and notations*

Let Γ be a zero sum game for two players Alice and Bernard such that:

– the sets $A = \{a_1, a_2, \ldots, a_m\}$ and $B = \{b_1, b_2, \ldots, b_n\}$ of possible deterministic strategies for Alice and Bernard are finite;

– the winnings for Alice when she applies strategy a_i and when Bernard applies strategy b_j, are denoted by the element $M_{i,j}$ of the matrix M.

The set of probabilistic strategies (or mixed strategies) is denoted by $\mathcal{A}$ for Alice, $\mathcal{B}$ for Bernard. These probabilistic strategies are obtained by combining the deterministic strategies (or pure strategies) according to a probability vector:

$$\begin{aligned}
\mathcal{A} = \quad & \{\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_m) \in [0,1]^m \text{ such that } \textstyle\sum_{i=1}^m \alpha_i = 1 \\
& \text{and the strategy } a_i \text{ is used with probability } \alpha_i\ \} \\
\mathcal{B} = \quad & \{\beta = (\beta_1, \beta_2, \ldots, \beta_n) \in [0,1]^n \text{ such that } \textstyle\sum_{j=1}^n \beta_j = 1 \\
& \text{and the strategy } b_j \text{ is used with probability } \beta_i\}
\end{aligned}$$

Of course, the pure strategies are included in the set of mixed strategies, as the probability vectors for which the full weight is on one component: a_1 corresponds to the mixed strategy of the probability vector $(1, 0, \ldots, 0)$. A mixed strategy is a linear combination of pure strategies: $\alpha = \alpha_1 a_1 + \alpha_2 a_2 + \ldots + \alpha_n a_n$.

COMMENT 2.1.– The performance of a mixed strategy α for Alice against a mixed strategy β for Bernard is calculated by the following formula:

$$\alpha^T M \beta = \sum_{i=1}^{m} \sum_{j=1}^{n} \alpha_i M_{i,j} \beta_j$$

A couple of strategies (α^*, β^*) is a *Nash equilibrium* of the game if, for all strategies α and β, $\alpha^T M \beta^* \leqslant {\alpha^*}^T M \beta^* \leqslant {\alpha^*}^T M \beta$. These configurations have the specificity that each player reduces his winnings if he is the only one to change strategy. For certain problems, there is a Nash equilibrium among the pure strategies. Von Neumann's minimax theorem shows that there is always a Nash equilibrium among the mixed strategies. Loomis's lemma shows that this equilibrium is attained by a couple formed from a pure strategy and a mixed strategy. The Yao principle is a reformulation of the minimax theorem which shows that the average complexity of the best deterministic algorithm over the worst distribution is the worst-case complexity of the best probabilistic algorithm. These results are proved in the following sections.

2.2.2. *Minimax theorem*

The minimax theorem is proved using the fixed point theorem, and lemmas 2.1 and 2.3. The lemmas are proved here; for the proof of the fixed point theorem, refer to [BOR 98, p. 112]. The proof of the minimax theorem introduced here is drawn from the one presented by Borodin and El-Yaniv [BOR 98], but slightly extended for clarification. Lemma 2.1 is a basic result used to show the double inequality of the minimax theorem (theorem 2.2).

LEMMA 2.1.– *The existence of $\tilde{\alpha}$ and $\tilde{\beta}$. Let ϕ and ψ be defined in $\mathbb{R}^m$ and $\mathbb{R}^n$ by:*

$$\phi(\alpha) = \sup_\beta \alpha^T M\beta \text{ and } \psi(\beta) = \inf_\alpha \alpha^T M\beta$$

Thus:

1) $\phi(\alpha) = \max_\beta \alpha^T M\beta$ and $\psi(\beta) = \min_\alpha \alpha^T M\beta$.

2) There are mixed strategies $\tilde{\alpha}$ for Alice and $\tilde{\beta}$ for Bernard such that ϕ reaches its minimum in $\tilde{\alpha}$ and ψ reaches its maximum in $\tilde{\beta}$.

Proof. The first point follows on from the bilinearity of M. If α is fixed, the function which associates $\alpha^T M\beta$ with β is linear, and thus iscontinuous, and, $\mathcal{B}$ being compact, it reaches its upper bound on $\mathcal{B}$ (theorem 29 in [SCH 81]): $\phi(\alpha) = \sup_\beta \alpha^T M\beta = \max_\beta \alpha^T M\beta$. The same logic applies to ψ.

To demonstrate this second point, it is sufficient to show that ϕ and ψ are continuous: the existence of $\tilde{\alpha}$ and $\tilde{\beta}$ is implied by the compactness of $\mathcal{A}$ and $\mathcal{B}$. To show the continuity of ϕ (the reasoning is equally valid for ψ), let $\alpha \in \mathcal{A}$ and $\varepsilon = (\varepsilon_\bullet, \varepsilon_\bullet, \ldots, \varepsilon_m)$:

$$\begin{aligned}\phi(\alpha+\varepsilon) &= \max_\beta(\alpha+\varepsilon)^T M\beta\\ &\leqslant \max_\beta \alpha^T M\beta + \max_\beta \varepsilon^T M\beta\\ &= \phi(\alpha) + \max_\beta \varepsilon^T M\beta.\end{aligned}$$

ut for any norm $|.|$, $|\varepsilon^T M\beta| \leqslant |\varepsilon| \times |M\beta|$. Since $\mathcal{B}$ is of finite dimension n, $|M\beta|$ s finite, and $|\varepsilon^T M\beta|$ tends towards zero when $|\varepsilon|$ tends towards zero. Therefore m$_{|\varepsilon|\to\bullet}$ $(\phi(\alpha+\varepsilon) - \phi(\alpha)) = 0$, and ϕ is continuous in α. ∎

The minimax theorem states that $\min_\alpha \max_\beta \alpha^T M\beta$ is equal to ax$_\beta \min_\alpha \alpha^T M\beta$. This is proved by showing a double inequality, the first of which

is valid even for pure strategies. Lemma 2.2 below is the "easy" part of the minimax theorem.

LEMMA 2.2.– *Simple inequality.*

$$\min_\alpha \max_\beta \alpha^T M\beta \geqslant \max_\beta \min_\alpha \alpha^T M\beta$$

Proof. Let $\tilde{\alpha}$ and $\tilde{\beta}$ be expressed by lemma 2.1:

$$\min_\alpha \max_\beta \alpha^T M\beta = \max_\beta \tilde{\alpha}^T M\beta \geqslant \tilde{\alpha}^T M\tilde{\beta} \geqslant \min_\alpha \alpha^T M\tilde{\beta} = \max_\beta \min_\alpha \alpha^T M\beta$$

∎

The lemma below is used to prove the minimax theorem.

LEMMA 2.3.– *Minimax lemma. In the game Γ the following two conditions are equivalent:*

(i) $\min_\alpha \max_\beta \alpha^T M\beta = \max_\beta \min_\alpha \alpha^T M\beta$.

(ii) There is a real number v and mixed strategies $\tilde{\alpha}$ and $\tilde{\beta}$ such that:

$\tilde{\alpha}^T M b_j \leqslant v \quad \forall j = 1, 2, \ldots, n$

$a_i^T M\tilde{\beta} \geqslant v \quad \forall i = 1, 2, \ldots, m$

Proof. $(ii) \Rightarrow (i)$ Let us assume the existence of the real number v and mixed strategies $\tilde{\alpha}$ and $\tilde{\beta}$ of (ii): by the linearity of M, for all $\beta \in \mathcal{B}$, we obtain $\tilde{\alpha}^T M\beta \leqslant v$. Therefore, in particular,

$$\max_\beta \tilde{\alpha}^T M\beta \leqslant v$$

which implies by the minimum definition, $\min_\alpha \max_\beta \alpha^T M\beta \leqslant v$. The same applies for $\tilde{\beta}$, $v \geqslant \max_\beta \min_\alpha \alpha^T M\beta$. So $\min_\alpha \max_\beta \alpha^T M\beta \leqslant v \leqslant \max_\beta \min_\alpha \alpha^T M\beta$. Moreover, lemma 2.2 implies the inverse inequality:

$$\min_\alpha \max_\beta \alpha^T M\beta \geqslant \max_\beta \min_\alpha \alpha^T M\beta$$

hence point (i).

$(i) \Rightarrow (ii)$ Let us assume that we have the equality (i). Let $v = \min_\alpha \max_\beta \alpha^T M\beta$. Using lemma 2.1, $\tilde{\alpha} \in \mathcal{A}$ exists such that $\max_\beta \tilde{\alpha}^T M\beta = v$. By definition, $\forall \beta \in \mathcal{B}$ $\tilde{\alpha}^T M\beta \leqslant v$, and the inequality $\tilde{\alpha}^T M b_j \leqslant v$ of (ii) is confirmed for all values of j from 1 to n. In the same way, there is a $\tilde{\beta} \in \mathcal{B}$ such that $\min_\alpha \alpha^T M\tilde{\beta} = v$, therefore $\forall \alpha \in \mathcal{A}$ $\alpha^T M\tilde{\beta} \geqslant v$, and as a consequence the inequality $a_i^T M\tilde{\beta} \leqslant v$ of (ii) is confirmed for all values of i from 1 to m. ∎

The proof of the minimax theorem given here relies on Brouwer's fixed point theorem.

THEOREM 2.1.– *Brouwer's fixed point [BOR 98, p. 112]. Let X be a compact and convex set in $\mathbb{R}^n$. Any continuous function $\phi : X \to X$ allows a fixed point $x \in X$ such that $\phi(x) = x$.*

THEOREM 2.2.– *von Neumann's minimax theorem. The game Γ is defined by the matrix M:*

$$\min_\alpha \max_\beta \alpha^T M\beta = \max_\beta \min_\alpha \alpha^T M\beta$$

Proof. Given the mixed strategies of each player α and β, let:

$$\begin{array}{lllll} p_i & = & p_i(\alpha,\beta) & = & a_i^T M\beta - \alpha^T M\beta \quad \forall i = 1,\dots,m \\ q_j & = & q_j(\alpha,\beta) & = & \alpha^T M\beta - \alpha^T M b_j \quad \forall j = 1,\dots,n \end{array}$$

Let the function Φ be: $\mathcal{A} * \mathcal{B} \to \mathcal{A} * \mathcal{B}$ such that each couple (α,β) matches the couple of mixed strategies (ξ,η) expressed by:

$$\forall i = 1,\dots,m \quad \xi_i = \frac{\alpha_i + \max\{p_i, 0\}}{1 + \sum_{k\bullet}^{n} \max\{p_k, 0\}}$$

$$\forall j = 1,\dots,n \quad \eta_j = \frac{\beta_j + \max\{q_j, 0\}}{1 + \sum_{k\bullet}^{m} \max\{q_k, 0\}}$$

The function Φ is the sum of continuous functions: it is continuous. $\mathcal{A}$ and $\mathcal{B}$ are compact and convex vector spaces. Hence theorem 2.1 implies that Φ allows a fixed point $(\tilde\alpha,\tilde\beta)$. Let p_i be considered at the point $(\tilde\alpha,\tilde\beta)$: $p_i = p_i(\tilde\alpha,\tilde\beta)$ for all i. We have:

$$\begin{aligned} \sum_i \tilde\alpha_i p_i &= \sum_i \tilde\alpha_i (a_i^T M\tilde\beta - \tilde\alpha^T M\tilde\beta) \\ &= (\sum_i \tilde\alpha_i a_i)^T M\tilde\beta - (\sum_i \tilde\alpha_i)\tilde\alpha^T M\tilde\beta \\ &= \tilde\alpha^T M\tilde\beta - \tilde\alpha^T M\tilde\beta \\ &= 0 \end{aligned}$$

The terms of a zero sum are either all zero, or include at least one negative and one positive term:

– If the terms are all zero, for all i $\tilde\alpha_i = 0$ or $p_i = 0$. Let us assume $p_i \neq 0$. From the fixed point definition $\tilde\alpha_i \sum_{k\bullet}^{n} \max\{p_k, 0\} = \max\{p_i, 0\}$: we have $\max(p_i, 0) = 0$ and $p_i \leqslant 0$. So $p_i \leqslant 0$ for all i.

– If not all the terms are zero, there is a strictly negative term $\tilde\alpha_i p_i$. $\tilde\alpha$ being a distribution of probability, $\tilde\alpha_i$ is positive, which implies that p_i is strictly negative. From the fixed point definition, $\tilde\alpha_i \sum_{k\bullet}^{n} \max\{p_k, 0\} = \max\{p_i, 0\}$, which is zero since $p_i < 0$. The terms of the zero sum of positive terms $\max\{p_k, 0\}$ are all zero, so $p_k \leqslant 0$ for all k.

The q_j are all shown to be negative or zero in a similar way. The point (ii) of lemma 2.3 is confirmed by noting that $v = \tilde\alpha^T M\tilde\beta$, which proves the theorem. ∎

2.2.3. *The Loomis lemma and the Yao principle*

The Loomis lemma is the essential argument for extending the minimax theorem to the Yao principle. The proof given here is the one given by Borodin and El-Yaniv [BOR 98].

LEMMA 2.4.– *The Loomis lemma [BOR 98, Lemma 8.2]. Given a fixed mixed strategy α for Alice, there is an optimal deterministic strategy b_j for Bernard when Alice applies the strategy α:*

$$\max_\beta \alpha^T M\beta = \alpha^T M b_j$$

In the same way, given a fixed mixed strategy β for Bernard, there is an optimal deterministic strategy a_i for Alice when Bernard applies the strategy β:

$$\min_\alpha \alpha^T M\beta = a_i^T M\beta$$

Proof. For a fixed α, $\alpha^T M$ is a linear function on $\mathcal{B}$, and for this reason reaches its maximum in at least one pure strategy b_j, corresponding to a maximum coefficient of $\alpha^T M$. Therefore, $\max_\beta \alpha^T M\beta = \alpha^T M b_j$. The same reasoning holds for Bernard. ∎

The equality of the minimax theorem can be rewritten by applying the Loomis lemma to optimal strategies $\tilde\alpha$ and $\tilde\beta$:

$$\min_\alpha \max_j \alpha^T M b_j = \max_\beta \min_i a_i^T M\beta \qquad [2.1]$$

These game theory results are applied to the analysis of algorithmic complexity. Let B be the *finite* set of instances of size s, $\mathcal{B}$ the set of random distributions over these instances, A a *finite* set of deterministic algorithms that solve the problem for a fixed data size s, and $\mathcal{A}$ the set of probabilistic algorithms defined by a distribution of probability over A.

Thus $\alpha^T M b_j$ is the average complexity of the probabilistic algorithm $\alpha \in \mathcal{A}$ over the instance $b_j \in B$, and $\max_j \alpha^T M b_j$ is the average complexity of this algorithm over the worst instance. Symmetrically, $a_i^T M \beta$ is the average complexity of the deterministic algorithm $a_i \in A$ over the distribution of instances $\beta \in \mathcal{B}$ and $\max_\beta a_i^T M \beta$ is the complexity of this algorithm over the worst imaginable distribution of instances.

The following inequality, known as "Yao's inequality", is obtained by applying this interpretation to equation [2.1]. This inequality allows us to obtain lower bounds to the complexity of probabilistic algorithms.

THEOREM 2.3.– *Yao principle [YAO 77]. The complexity of the best deterministic algorithm over the worst distribution of instances $\beta \in \mathcal{B}$ is equal to the complexity of the best probabilistic algorithm over the worst instance:*

$$\max_\beta \min_i a_i^T M \beta = \min_\alpha \max_j \alpha^T M b_j$$

This approach allows us to obtain lower bounds for *finite* sets of deterministic algorithms over sets of finite instances. This is not a difficult bound to reach, for a finite number of instances, once the size of the instance has been fixed. For example, the computational model based on comparisons is reduced to the algorithms that do not carry out the same comparison twice. The number of such algorithms is polynomial in the number of elements to be compared. There is therefore a finite number of algorithms on instances of fixed size.

COMMENT 2.2.– The fact that $\mathcal{A}$ and $\mathcal{B}$ are of a finite dimension is crucial in all the extensions of the minimax principle [SIO 58].

EXAMPLE.– In the hidden coins problem, a single silver coin being uniformly hidden, any deterministic algorithm uncovers an average of $n/2$ cards. The Yao principle implies that the best Las Vegas algorithm uncovers on average $n/2$ cards for at least one configuration.

This factor of $\div$ is not very important, but more important differences in perfor-
nance appear between deterministic and probabilistic algorithms when complexity is
nalyzed more closely.

XAMPLE.– In the hidden coin problem, if half the coins are silver, and if they are
venly distributed, the best deterministic algorithm uncovers $\lceil n/2 \rceil$ coins in the worst
ase, while the best probabilistic algorithm uncovers less than 2 coins, on average, in
ne worst instance.

2.3. Elementary intersection problem

The analysis of the hidden coin problem can be adapted to other, less trivial problems, which can be decomposed as a conjunction of independent subproblems. The elementary intersection problem, which is relatively close to the hidden coins problem used throughout this chapter, is given as an applied example.

Formally, an instance of the elementary intersection problem is made up of an element x and k sorted tables, of respective sizes $n_\bullet \leqslant \ldots \leqslant n_k$. The problem therefore consists of deciding whether or not x belongs to the intersection of the tables, that is whether there is a table that does not contain x. The analogy with the hidden coins problem can be seen if we make a copper coin correspond to the tables containing x, and a silver coin to the tables not containing x. As for the hidden coin problem, the instance is harder if almost all the tables contain x, both for the deterministic and the probabilistic algorithms, and a probabilistic algorithm will end more quickly than a deterministic algorithm if only half the tables contain x.

We will show here how to obtain a lower bound on the complexity of the elementary intersection problem using the Yao principle, and how to obtain an upper bound by analyzing a simple randomized algorithm.

2.3.1. *Upper bound*

The complexity of any probabilistic algorithm for a given problem forms an upper bound on the probabilistic complexity of this problem. The bound is all the better since the complexity of the algorithm is reduced. By definition, this complexity is equal to the average of the complexities of the deterministic algorithms making up the probabilistic algorithm on a fixed worst instance: $\max_I \sum_i p_i C(A_i, I)$. This corresponds to the last example in section 2.1.2, in the case where half the coins are made of silver.

To calculate this complexity, we draw upon the Yao principle, adapting the randomization of the algorithm on the instance, in such a way as to define a worst distribution of instances, from the worst instance: then the complexity in the worst case of the probabilistic algorithm corresponds to the average complexity of a deterministic algorithm over a worst distribution.

EXAMPLE.– In the example of the hidden coins problem, any probabilistic algorithm can be defined as a distribution $(p_i)_{i \in \{\bullet, \ldots, n_\bullet\}}$ over the permutations of $\{1, \ldots, n\}$. For any instance I, and in particular for the worst instance, $(p_i)_{i \in \{\bullet, \ldots, n_\bullet\}}$ defines a distribution over the permutations of I. The complexity of a probabilistic algorithm then corresponds to the average complexity of any deterministic algorithm on this distribution.

The complexity of any probabilistic algorithm that solves the elementary intersection problem forms an upper bound to the probabilistic complexity of the problem.

Let us consider a deterministic algorithm that is looking for element x in each table using the dichotomous search algorithm. The cost of the search, in numbers of comparisons in a table of n_i elements, is $O(\log n_i)$. Thus this type of deterministic algorithm carries out at most $O(\sum_{i=1}^k \log n_i)$ comparisons.

Therefore, the probabilistic complexity of the elementary intersection problem of an instance of signature $(k, n_1, \dots, n_k)$ must be less than or equal to, give or take a multiplicative constant, $\sum_{i=1}^k \log n_i$. We will show that this upper bound corresponds to the upper bound, and that therefore this deterministic algorithm is optimal amongst all the deterministic or probabilistic algorithms that solve the elementary intersection problem.

2.3.2. *Lower bound*

To prove a lower bound on the worst-case complexity of any *deterministic* algorithm, it is sufficient to define an opponent's strategy, which constructs, for each deterministic algorithm, its worst instance. Such a method provides a lower bound of $\sum_{i=1}^k \log n_i$, which proves that the type of algorithm described in the previous section is optimal among the deterministic algorithms, but does not prove anything about probabilistic algorithms.

To prove a lower bound on the worst-case complexity of any *probabilistic* algorithm, we use the Yao principle, which allows us to obtain this bound from a worst distribution for all the deterministic algorithms. The following lemma defines such a worst distribution.

LEMMA 2.5.– *For any set of integers $k > 1$, $n_1, \dots, n_k > 0$, there is a distribution of instances $(x, A_1, \dots, A_k)$ of signature $(k, n_1, \dots, n_k)$ such that any deterministic algorithm that decides whether x is in the intersection $A_1 \cap \dots \cap A_k$ carries out on average at least $\Omega(\sum_{i=1}^k \log n_i)$ comparisons.*

The distribution is defined by choosing a table A_w where x may be absent with probability $\log n_i / \sum \log n_i$ for each table A_i; for each other table, we choose a position where x may be placed with probability $1/n_i$ for each table A_i. Any algorithm looking for x will find it in $k/2$ tables on average before finding the table A_w that does not contain x, and will carry out, in each of these tables, on average $\Omega(\sum_{i=1}^k \log n_i)$ comparisons. The rigorous proof of the lower bound is not relevant to us, and is given elsewhere [BAR 08].

By directly applying the Yao principle we obtain the desired lower bound.

THEOREM 2.4.– *For any set of integers $k > 1$, $n_1, \dots, n_k > 0$, and for any probabilistic algorithm A that solves the elementary intersection problem, there is an instance $(x, A_1, \dots, A_k)$ of signature $(k, n_1, \dots, n_k)$ such that A carries out on average at least $\Omega(\sum_{i=1}^k \log n_i)$ comparisons on the instance $(x, A_1, \dots, A_k)$.*

The probabilistic complexity of the elementary intersection problem is therefore greater than or equal to $\sum_{i=1}^k \log n_i$, give or take a multiplicative constant.

2.3.3. *Probabilistic complexity*

The previous results give lower and upper bounds on the probabilistic complexity accurate to one multiplicative constant. This is sufficient to know the order of magnitude of the probabilistic complexity of the elementary intersection problem, and to prove that the type of deterministic algorithm proposed in section 2.3.1 is optimal.

2.4. Conclusion

The analysis of probabilistic algorithms is not generally considered as combinatorial optimization, while heuristics, particular probabilistic algorithms, are often essential to solve **NP**-difficult problems, and in particular for optimization problems. This is certainly because in practice it is difficult to analyze the algorithms used, and rare to be able to prove close lower and upper bounds. Let us hope that this chapter helps to bridge the gap between the theoretical analysis of probabilistic complexity and the study of heuristics.

2.5. Bibliography

[BAR 08] BARBAY J., KENYON C., "Alternation and redundancy analysis of the intersection problem", *ACM Trans. Algorithms*, vol. 4, num. 1, p. 1–18, 2008.

[BOR 98] BORODIN A., EL-YANIV R., *Online Computation and Competitive Analysis*, Cambridge University Press, Cambridge, 1998.

[KNU 73] KNUTH D.E., *The Art of Computer Programming, vol. 3: Sorting and Searching*, Addison-Wesley, Reading, MA, 1973.

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

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

[SCH 81] SCHWARTZ L., *Cours d'analyse*, Hermann, Paris, 1981.

[SIO 58] SION M., "On general Minimax theorems", *Pacific Journal of Mathematics*, p. 171–176, 1958.

[YAO 77] YAO A.C., "Probabilistic computations: Toward a unified measure of complexity", *Proc. FOCS'77*, p. 222–227, 1977.

PART II

Classical Solution Methods