Chapter 1
Basic Concepts

Ferrante Neri and Carlos Cotta

1.1 What is Optimization?

In every day life, we always have to make decisions, e.g. the path to choose in order to go back home from work, the brand of milk in a supermarket, whether to watch football or a movie in TV, etc. Some of these choices appear to us obvious while some other choices require some thinking. Regardless the context, decisions are usually made in order to reach a certain goal or satisfy a given necessity. For example, in the case of going back home from work, a reasonable goal would be to choose a path which leads us back home in the shortest possible time. Let us assume that the path should be performed by walking. In this case, the solution for the problem is likely to be the shortest path. This would be a simple optimization problem. If the goal would be to be at home at the earliest after having bought something in the city center, e.g. a visit a shop, we have to exclude some of the possible paths. More specifically, we have to take into account only the paths which pass through the shop. The path having the latter features are said to be feasible while all the others are infeasible. The newly stated problem is a constrained optimization problem. If an additional goal, beside being back at home in the shortest possible time, is to take the opportunity for having some physical activities by means of a long walk, two conflicting objectives must be taken into account and a compromise must be accepted (e.g. a not too long path to be at home reasonably early and a not too short path to have at least some physical activity). Due to the presence of two simultaneous and conflicting goals, the latter is a multi-objective optimization problem.

Ferrante Neri
Department of Mathematical Information Technology, P.O. Box 35 (Agora), 40014, University of Jyväskylä, Finland, e-mail: ferrante.neri@jyu.fi

Carlos Cotta
Dept. de Lenguajes y Ciencias de la Computación. Universidad de Málaga, Campus de Teatinos, 29071 Málaga, Spain, e-mail: ccottap@lcc.uma.es

More mathematically, let us consider a solution x, i.e. a vector of n design variables $(x_1, x_2, \ldots, x_i, \ldots, x_n)$. Each of the design variable x_i can take values from a domain $\mathscr{D}_i$ (e.g., an interval $[x_i^L, x_i^U]$ if variables are continuous, or a certain discrete collection of values otherwise). The Cartesian product of these domains for each design variable is called the decision space $\mathscr{D}$. Let us consider a set of functions $f_1, f_2, \ldots, f_m$ defined in $\mathscr{D}$ and returning real values. Under these conditions, the most general statement of an optimization problems is given by the following formulas:

$$\begin{array}{cccc} Maximize/Minimize & & f_m & m = 1,2,\ldots,M \\ subject-to & & g_j(x) \leqslant 0 & j = 1,2,\ldots,J \\ & & h_k(x) = 0 & k = 1,2,\ldots,K \\ & & x_i^L \leqslant x_i \leqslant x_i^U & i = 1,2,\ldots,n \end{array} \tag{1.1}$$

where g_j and h_k are inequality and equality constraints, respectively.

From the definition above, we can easily see that if $m = 1$ the problem is single-objective, while for $m > 1$ the problem is multi-objective. The presence/absence of the functions g_j and h_k make the problem more or less severely constrained. Finally, the continuous or combinatorial nature of the problem is given by the fact that $\mathscr{D}$ is a discrete or dense set. In other words, all the problems considered in this book can be considered as specific cases of the general definition in equations (1.1).

In the continuous case, for each m the detection of a maximum or minimum point requires the detection of those points characterized by a null gradient, i.e.:

$$\nabla f = \begin{bmatrix} \frac{\partial}{\partial x_1} \\ \frac{\partial}{\partial x_2} \\ \ldots \\ \frac{\partial}{\partial x_n} \end{bmatrix} = \bar{0} \tag{1.2}$$

In general, in a multidimensional continuous decision space $\mathscr{D}$, there are several points satisfying the condition in eq. (1.2). Some of these points are minima, some are maxima and some are saddle points. While solving an optimization problem, e.g., a minimization, it is fundamental to distinguish the three kinds of point. In order to distinguish them, the determinant of the Hessian matrix should be discussed. More specifically, the Hessian matrix is:

$$H(x) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial}{\partial x_1}\frac{\partial}{\partial x_2} f & \cdots & \frac{\partial}{\partial x_1}\frac{\partial}{\partial x_n} f \\ \frac{\partial}{\partial x_2}\frac{\partial}{\partial x_1} f & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\partial}{\partial x_n}\frac{\partial}{\partial x_1} f & \cdots & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix} \tag{1.3}$$

In order to check whether a point x_0 is a minimum, a maximum, or a saddle point, the determinant Δ of the Hessian matrix must be checked. If

$$\Delta > 0 \text{ and } \frac{\partial^2 f}{\partial x_0^2} > 0, \tag{1.4}$$

x_0 is a local minimum; if

$$\Delta > 0 \text{ and } \frac{\partial^2 f}{\partial x_0^2} < 0, \tag{1.5}$$

x_0 is a local maximum; if $\Delta < 0$, x_0 is a saddle point.

The situation is much more subtle in the case of combinatorial domains, in which the notion of locality for optima is associated to a particular definition of neighborhood among the discrete elements in $\mathscr{D}$.

1.2 Optimization Can Be Hard

In real-world applications, it is usually not so important to detect local optima. The global optimum is usually of interest for engineers and practitioners. Thus, in principle, all the null gradient points should be detected and analyzed before selecting the global optimum. In practical problems, this set of operations is not always possible as often the objective function is not differentiable within the entire decision space, or is not even available in an explicit analytical form (being e.g. a procedure, a simulation, or an experiment measurement). In addition, it must be remarked that from en engineering/application viewpoint it is fundamental to detect a solution which displays a high performance and is usually irrelevant whether or not this solution corresponds to a null gradient.

When regarded from a computational perspective, the above ideas can be characterized in terms of computational complexity. Assuming a certain computational framework (e.g., Turing machines), it is possible to measure the amount of resources (time or space to give two distinguished examples) that a certain algorithm requires in order to fulfill its objective, e.g., finding the global optimum for a certain optimization problem. By analyzing the growth of such resource consumption in terms of the size of the problem instance considered it is possible to define complexity classes of problems. More precisely, we can denote as $\mathbf{REC}(f(n))$ the class of problems for which there exists an algorithm (not necessarily the same algorithm for all problems in the class) that solves any instance of size n using at most $f(n)$ units of resource REC. It is customary –yet sometimes unrealistic– to consider that a problem is tractable if it can be solved in polynomial time, i.e., if it belongs to class $\mathbf{TIME}(n^k)$ for some fixed k. In case of decision problems (those for which a yes/no response is sought), this definition amounts to the well-known class P.

Using the notion of reduction (an *efficient*[1] mechanism for transforming an instance of problem A into an instance of problem A'), we can define a problem A as **C**-hard if any problem in class **C** can be reduced to A (hence A is at least as hard to

[1] The notion of efficiency here refers to the particular complexity class under consideration, e.g., polynomial time when studying classes in the polynomial hierarchy [1].

solve as any problem in **C**). If a problem is **C**-hard and also belongs to class **C**, it is termed **C**-complete. Problems complete for a class are useful in characterizing the actual complexity of the class.

It turns out that many interesting problems are NP-complete when their decision version is considered, that is, they can be solved in polynomial time by a non-deterministic Turing machine (or alternatively, a yes-solution can be verified in polynomial time). Clearly, class P is a subset of class NP and, although yet unproven, it is widely believed that P is a proper subset of NP, i.e., P$\neq$NP. This means that no efficient –polynomial-time– algorithm is known to solve the problem to optimality. Furthermore, many real-world problems can also be shown to be hard to approximate, i.e., there exist no efficient algorithm capable of providing solutions whose quality is guaranteed to be within a certain distance of the optimum (several complexity classes can be defined in terms of the approximation ratios attainable [2]).

This complexity barrier can be dealt with using two different (and complementary) approaches. The first one is the use of parameterized complexity techniques. These techniques try to factor out some part of the problem input as a parameter k, and provide $\mathbf{TIME}(f(k)n^c)$ (where c is a constant that does not depend on the parameter k and $f(\cdot)$ is an arbitrary function of k) algorithms for these problems. Assuming realistic instances of the problems would just exhibit low parameter values k, these algorithms turn out to provide efficient solutions to the problems under consideration (which are thus termed fixed-parameter tractable). The second potential approach is the use of metaheuristics, as discussed next.

1.3 Using Metaheuristics

When hypotheses on the optimization problem cannot be made, a general purpose optimization algorithm/procedure must be implemented for solving the problem or at least detecting some solutions with a high performance. General purpose algorithms are usually referred as metaheuristics from the ancient Greek words μετα and ευρισκω , i.e., literally "*I search beyond*" or more generally "*beyond the search*", in the sense that the search can be done at an abstract level to the result of another search procedure.

Metaheuristics have been developed during the last decades jointly with the progress of computational hardware and, nowadays, there exists a huge variety of general purpose optimization algorithms. Some of them, gets inspiration from the nature, e.g. evolutionary principles, physical phenomena, animal behaviour, etc., in order to tackle the problem. These nature inspired methods are also known as Computational Intelligence Optimization algorithms since they use Computational Intelligence (CI) to face optimization problems. Traditionally, CI was identified as subject including Fuzzy Systems, Neural Networks and Evolutionary Computation. This definition appears today too restrictive and outdated, since other recently defined algorithmic structures, such as Swarm Intelligence, can also fit within CI.

Amongst these emergent metaheuristics or, if we prefer, CI optimization algorithms, Memetic Algorithms (MAs) represent a successful story which developed during the last two decades and are year after year becoming an important CI paradigm which allows the solution of complex optimization problems.

This book attempts to explain in depth the algorithmic and implementation aspects of this paradigm, its variations in optimization problems under specific circumstances, some implementation in specific application domains, and finally the historical context where the terms have been coined and the early implementations have been performed. More specifically, this book is divided into four parts, the first about basic concepts and algorithmic components, the second is about specific MA implementations and problems, and the third part is about MA applications. Finally, the last part gives some historical background and biographical notes regarding the earliest definition of MAs.

Acknowledgements F. Neri is supported by the Academy of Finland, Akatemiatutkija 130600, Algorithmic Design Issues in Memetic Computing. C. Cotta is partially supported by Spanish MICINN under project NEMESIS (TIN2008-05941) and by Junta de Andalucía under project TIC-6083.

References

1. Papadimitriou C (1994) Computational Complexity. Addison Wesley
2. Vazirani V (2001) Approximation Algorithms. Springer-Verlag, Berlin

Index