PART 1

Optimization and Artificial Evolution

Chapter 1

Optimization: State of the Art

In this chapter, we present the methodological principles involved in optimization, before introducing the main optimization methods used in an industrial context.

1.1. Methodological principles in optimization

This section presents the main characteristics of modeling of industrial optimization problems.

1.1.1. *Introduction*

When faced with a real optimization problem, we must analyze the problem in a precise manner in order to choose the best method to use. Real optimization problems correspond to needs observed in industrial or operational contexts, and aim to improve the performance of an economic process connected with an operational company or management organization. In practice, these problems are identified by domain experts who wish to develop an

optimization principle in order to improve the performance of a system.

1.1.2. *Modeling*

The first stage in the optimization process consists of modeling the real problem using a mathematical abstraction that is as efficient as possible (see Figure 1.1). Using this abstraction, it is possible to develop solution algorithms that can be executed on a computer. This optimization process produces a set of solution points, which can then be implemented in the real world. In the past, there were few choices of optimization algorithms and it was necessary to use models that were somewhat different from reality but for which solution methods existed. Therefore, the solutions produced could be different from the true solution in the real world. A classic example involves linear programming (LP) for which we have efficient solution algorithms, but which requires linear modeling of the problem.

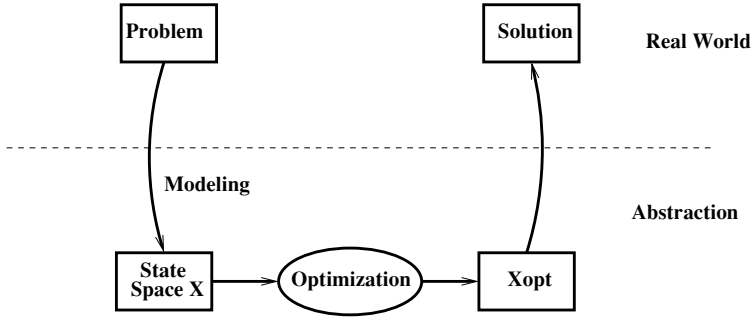

Figure 1.1. *Modeling process*

The modeling stage, then, consists of characterizing the state space and the objective space.

1.1.2.1. *State space*

The state space represents the set of parameters of the system upon which we may act in order to optimize one (or more) objective(s). Examination of the properties of the state space then helps us in choosing a suitable optimization method.

In most industrial optimization problems, the variables of the state space must remain within a subdomain defined by a set of constraints. We obtain the following general model:

$$\begin{cases} \min y = f(\vec{x}) \\ \vec{x}_{opt} \in \mathcal{A} \subset \mathcal{X} \end{cases}$$

where $\mathcal{X}$ is the state space and $\mathcal{A}$ is the feasible space bounded by the constraints. By studying the properties of $\mathcal{X}$ and $\mathcal{A}$, we can determine certain characteristics of the solution algorithm. Thus, the properties of connectivity and convexity of the admissible domain $\mathcal{A}$ are extremely important in the right choice of an optimization method. Within the group of convex state spaces, there is an extremely interesting subclass for which the admissible domain is bounded by a set of hyperplanes making up a polytope. If, moreover, the criterion is linear, then we have a linear optimization problem for which we may use the Danzig simplex method, for example. In the same way, the properties of connectivity of the state space determine whether or not it will be necessary to allow the optimization method to violate constraints in order to transit from one component to another to finally reach the optimum solution.

Based on the nature of state variables, we may classify the industrial optimization problems into three categories:

1) *continuous problem*

$$\mathcal{X} = \mathcal{U}_1 \times \mathcal{U}_2 \times ... \times \mathcal{U}_m$$
$$\mathcal{U}_i \subset \mathbb{R}\, i = 1, 2, .., m$$

We talk of optimization in a functional space when m is infinite (as in the case of trajectory optimization).

2) *discrete problem*

$$\mathcal{X} = \mathcal{I}_1 \times \mathcal{I}_2 \times ... \times \mathcal{I}_n$$
$$\mathcal{I}_i \subset \mathbb{Z}\, i = 1, 2, .., n$$

3) *mixed problem*

$$\mathcal{X} = \mathcal{U}_1 \times \mathcal{U}_2 \times ... \times \mathcal{U}_m \times \mathcal{I}_1 \times \mathcal{I}_2 \times ... \times \mathcal{I}_n$$

Mixed problems are the most difficult of the three classes to work with.

For certain problems, it is possible to voluntarily limit the feasible space by eliminating *a priori* the subdomains which we know will not contain the optimum. This restriction may be more or less accurate based on the information available for the specific problem. If, for example, we wish to optimize a firing angle to maximize the distance achieved by a projectile, we clearly need only to look at angles between 0 and $\frac{\pi}{2}$ (initial firing angle in relation to the ground). This type of reasoning is extremely useful in optimization algorithms as it avoids unnecessary exploration of some areas of the state space. It arises from a more general concept, which may be summarized as follows: the more we can bring information to the solution algorithm, the better this algorithm will perform.

One very important point characterizing the state space is its dimension. Generally, the higher the dimension n of $\mathcal{X}$, the harder it will be to find the optimum.

Optimization algorithms are implemented in a programming language which runs on a computer. Computer memory is always limited, in spite of significant progress in the domain. Thus, if the memory representation of a point in the state space is large, certain methods will be hindered by

this limitation, in particular those that use populations of points of the state space (e.g. genetic algorithms).

Finally, we may face problems for which the state space has an infinite dimension. This is the case for trajectory optimization problems, where simple sampling should be avoided as it produces moderate results. It is preferable to use suitable decomposition bases and return to an optimization principle in a finite space by controlling the decomposition coefficients. The major challenge with this type of approach is to find a base suited to the real problem under consideration.

1.1.2.2. *Objective space*

The objective space represents the set of criteria that we wish to optimize. Based on the dimension of this space, we can identify two classes of problems:

– Mono-objective problems: this is the simplest case, insofar as a single criterion needs to be optimized and enables a total order relationship between points of the state space in terms of the criterion. The objective function is thus a function of $\mathbb{R}^n$ in $\mathbb{R}$.

– Multi-objective problems: in this case, we need to optimize several criteria, associated with each point of the state space, simultaneously. The most critical aspect of such problems is linked to the loss of the total order relationship between the solutions. Effectively, the objective function is now a function of $\mathbb{R}^n$ in $\mathbb{R}^m$, where m is the dimension of the objective space. Let $\vec{x}_a$ and $\vec{x}_b$ be two points of the state space, with which we associate the objective vectors $\vec{y}_a$ and $\vec{y}_b$ for which each component needs to be maximized (for example).

Finally, to simplify, let us consider $m = 2$ and the following three cases:

Case 1	Case 2	Case 3
$\vec{y}_a = (5,8)^T$	$\vec{y}_a = (5,8)^T$	$\vec{y}_a = (5,8)^T$
$\vec{y}_b = (3,4)^T$	$\vec{y}_b = (7,9)^T$	$\vec{y}_b = (6,3)^T$

In the first case, the two objectives of the solution “a” are better than those of solution “b”; we may say that solution “a” dominates solution “b”. The second case corresponds to the opposite situation: “b” dominates “a”. In the third case, however, it is impossible to identify a dominant solution as one of the objectives is better in each solution; we may say that “a” and “b” are mutually non-dominant. In the case of multi-objective optimization, we look for these non-dominant solutions which, are grouped into a set known as the Pareto front.

These solutions are then examined by an expert in the domain of application in order to identify the best solution for implementation in the real-world context.

In the case of mono-objective problems, we may also characterize the objective space in relation to the optima of the criterion. We can then distinguish between two types of optima:

1) *Global optimum:* $\vec{x}^*$

$$f(\vec{x}^*) \leq f(\vec{x}) \; \forall \vec{x} \in \mathcal{X}$$

$\mathcal{X}$ completes the state space.

2) *Local optimum:* $\tilde{x}$

$$f(\tilde{\vec{x}}) \leq f(\vec{x}) \; \forall \vec{x} \in \mathcal{V}(\tilde{\vec{x}})$$

$\mathcal{V}(\tilde{\vec{x}})$ vicinity of $\tilde{\vec{x}}$.

Within the context of industrial optimization problems, we seek to determine global optima and try to avoid being stuck on local optima.

The convexity of the criterion for optimization is thus a fundamental characteristic used for the right choice of an

optimization method. When dealing with a (strictly) convex problem, there is only one optimum; it is therefore possible to use a local method to identify this optimum. In the opposite case, this local method would determine a local optimum (often that is closest to the initial starting point).

For certain non-convex problems, the function to optimize may present several quasi-equivalent optima which we need to identify. In this case, we speak of a multi-mode function. This type of problem requires a multi-mode optimization method to extract these equivalent optima. These multi-mode problems are generally more difficult to work with.

The continuity of the objective function is also a determining factor in the development of an optimization principle. For continuous problems, it is easy to approximate the slope of the criterion and thus orient the method used for resolution. When the criterion slope is bounded throughout the state space, the state space is said to be Lipschitz in terms of gradient, which allows us to guarantee the convergence of certain optimization methods. In the opposite case, these methods can diverge strongly in zones of discontinuity. In the same way, if large zones of the state space present plateaus, these methods, which use slope for orientation, tend to be ineffective for these regions.

More generally, all optimization methods require variation in the criterion across the state space in order to be directed toward the optima. This is the principle of locality in optimization. One case in which this principle is not followed is a criterion which takes the form of a plateau, for which only certain points of the state space present isolated peaks (see Figure 1.2). In this example, no optimization method will be able to find these optima.

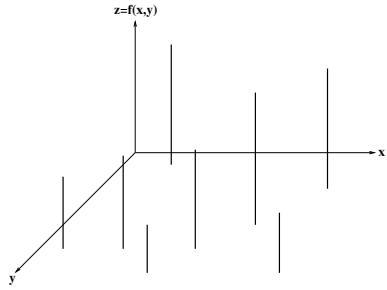

Figure 1.2. *Function for which the locality principle is not respected. Each vertical black line represents the peak of the function*

Another important point associated with an objective function is whether or not the function is bounded. This information allows us to define the termination criteria for certain stochastic optimization methods. These methods move through the state space in a random manner and do not ensure convergence to a global optimum for a given execution (proof of stochastic convergence is limited: the mathematical expectation of the set of solutions provided across multiple executions is equal to the global optimum of the function). Thus, if we know the value of the criterion at the optimum (and not its position in the state space), it is relatively easy to terminate a stochastic method by observing the value of the criterion for the current solution. If the value of the minima of a positive objective function is equal to zero, a stochastic method producing solutions with a value of the criterion close to zero is certain to be close to an optimum and may be terminated.

Consideration of the separability of the objective function also allows us to simplify the optimization algorithm. Let us take a function, $f(\vec{x})$, to minimize for the space $\mathcal{X}$. If $f(\vec{x}) = f\{g(\vec{x}), h(\vec{x})\}$ and if

$$\left\{ \begin{array}{l} min\ f(\vec{x}) \\ \vec{x} \in \mathcal{X} \end{array} \right. = f \left\{ \begin{array}{l} min\ g(\vec{x}) \\ \vec{x} \in \mathcal{X} \end{array} , \begin{array}{l} min\ h(\vec{x}) \\ \vec{x} \in \mathcal{X} \end{array} \right\}$$

then the function f is separable. This property allows us to independently optimize functions g and h, which may be simpler to process. The most advantageous case is when g and h are functions for which the state spaces are orthogonal subspaces of $\mathcal{X}$ (state space of function f) with smaller dimensions. This reduction of the dimension allows the resolution principle to be accelerated.

The principles of evaluation of the criterion also enable us to select more or less suitable resolution algorithms. We may distinguish among three different types of cases:

1) Criterion accessible in analytical form: this is the most "comfortable" case, but is unfortunately rare in the context of real problems. By cancelling the associated gradient, it is sometimes possible to obtain an analytical form of the optimum (textbook case).

2) Criterion evaluated through numerical computations using real data: this is the most frequent of the three cases in which we attempt to extract additional information to guide the algorithm (gradient, Hessian, etc.).

3) Criterion evaluated using a complex simulation process: in this case, where evaluation of the criterion is often costly in terms of resources, it is not possible to obtain additional information and we use methods that do not need a criterion value in order to converge.

For certain problems, we also need to take into account the fact that the evaluation of the criterion is affected by noise and select a method that is not affected too much by such noise.

Finally, certain problems present dynamic criterion landscapes which require optimum tracking techniques. The timescale for the evolution of the criterion must be placed in a relationship with the time needed for the optimization algorithm to reach the optimum. If these two characteristics

are of the same order of magnitude, we have an optimization problem with a dynamic criterion. Currently, only artificial evolution methods allow us to correctly solve this type of problem.

1.1.3. *Complexity*

The complexity of an optimization problem is linked to the number of operations needed to determine the optimum. In the case of combinatory optimization problems, we consider an instance of a problem of size n for which a resolution algorithm has been proposed. This algorithm will execute a number of operations K to process the problem. This number K is generally dependent on n. Based on the type of relationship between K and n, we can classify solution algorithms as follows:

Notation	Types of complexity
$O(1)$	constant complexity (independent of data size)
$O(\log(n))$	logarithmic complexity
$O(n)$	linear complexity
$O(n.\log(n))$	quasi-linear complexity
$O(n^2)$	quadratic complexity
$O(n^3)$	cubic complexity
$O(n^p)$	polynomial complexity
$O(n^p.\log(n))$	quasi-polynomial complexity
$O(2^n)$	exponential complexity
$O(n!)$	factorial complexity

In the same way, we classify optimization problems based on the best algorithms for their solution. Thus, we have a class of NP-hard problems, which cannot be solved using a known polynomial algorithm. An examination of the complexity associated with an optimization problem allows us to select an optimization method.

1.1.4. *Computation time*

Given that each optimization method requires a minimum computation time, it is important to be aware of the time we have available to produce a solution. As an example, in the context of an optimization problem concerning airspace sectorization, we have several months to produce a solution (6 months is needed before implementing a new sectorization). In the context of optimization of flight plans for a day, we have 24 h. To solve conflicts between aircraft, we have 3 min; finally, for a problem concerning satellite frequency allocations, we must produce a solution within 50 ms. In cases where this constraint is critical, we need to look for parallel methods. Two types of parallelism exist for methods using populations of points of the state space:

– Parallelization of criterion computation: this approach is advantageous in cases where the communication time between processors is low in relation to the time taken to compute the criterion.

– Islet parallelization: this allows effective parallel calculation, even in cases where criteria are computed rapidly.

1.1.5. *Conclusion*

When addressing an industrial optimization problem, we must respond to certain questions in order to create a suitable solution strategy:

– How can the problem be modeled?

- What objectives do we wish to optimize? (Mono-objective or multi-objective? Convex criterion? Linear? Quantified continuous? etc.)

- What parameters can (must) we act on? (Continuous or discrete state space? Large or infinite dimensions? etc.)

- What are the constraints? (Connected space? Convex space? Linear constraints? etc.)

– How much time do we have for the calculation?

– How much memory is needed for a point in the state space?

– How complex is the problem? (NP-hard?)

– Can we relate our problem to a known problem?

In the remainder of this chapter, we will discuss a number of solution principles used when dealing with industrial optimization problems.

1.2. Optimization algorithms

This section provides a concise overview of the main optimization methods by field of application.

1.2.1. *Introduction*

In what follows, the term “global optimization” will be used to refer to the search for global optima of the objective function. However, this term is somewhat ambiguous, as we often find the term “local search” in specialist literature, which refers to the search mechanism when it proceeds using successive neighbors. Thus, simulated annealing is a local search method (the tested solution is a neighbor of the current solution); from our perspective, this is a global optimization method (the method is, by principle, capable of determining the global optima of the objective function).

Optimization methods may be divided into two categories: local methods, which allow us to identify a local optimum, and global optimization methods, which are used to determine a global optimum.

Finally, depending on the mechanism used to move within the state space, we differentiate between deterministic and stochastic methods.

We will start with a description of local methods.

1.2.2. *Linear programming*

In the case of a linear problem, this method is able to determine a global optimum, but it cannot be applied to a general global optimization problem. For this reason, this method is classified as a local method.

LP is an extremely powerful operational research tool. A linear program is made up of a linear objective function and a set of constraints (equalities and/or inequalities) which are also linear. All linear programs may be written in the following canonical form:

$$\begin{array}{rrcl} & \max z = & \vec{c}^T.\vec{x} & \\ s.c & A\vec{x} & \leq & \vec{b} \\ & \vec{x} & \geq & \vec{0}, \end{array} \qquad [1.1]$$

where $\vec{c}$ and $\vec{x}$ are vectors of size n, $\vec{b}$ is a vector of size m and A is a matrix of size $m \times n$.

The feasible domain is then represented in the form of a polytope (simplex).

The simplex algorithm (presented for the first time by George Bernard Dantzig in 1947) allows us to solve LP problems by first creating a feasible solution, which is a vertex of a polytope, then moving along the edges of the polytope in order to reach vertices for which the value of the objective is increasingly high, until the optimum is reached (in the case of a maximization).

While this algorithm is effective in practice and guarantees that the optimum will be found, it does not always behave in a satisfactory manner in the worst cases. It is possible to create an LP for which the simplex method requires an exponential number of steps as a function of the size of the problem. Thus, the question of whether LP was an NP-complete or a polynomial problem remained unresolved for a number of years.

The first polynomial algorithm for LP was proposed by Leonid Khachiyan in 1979.

However, the practical effectiveness of Khachiyan's algorithm is disappointing, and the simplex algorithm nearly always performs better. Nevertheless, this result encouraged research into interior point methods. Unlike the simplex algorithm, which only considers the edges of the polytope defined by constraints, interior point methods operate inside the polytope.

In 1984, N. Karmarkar [KAR 84] developed the projective method. This was the first algorithm to be effective in both theory and practice. In the worst cases, its complexity is polynomial and experiments using practical problems have demonstrated that the method can reasonably be compared to the simplex algorithm.

1.2.3. *Nonlinear programming (NLP)*

1.2.3.1. *Methods of order zero*

The Nelder–Mead method is a local optimization algorithm developped by Nelder and Mead in 1965 [NEL 65]. This method uses the concept of the simplex, which is a polytope with $N + 1$ vertices in a space of N dimensions. Let N be the dimension of the state space. We therefore begin with a simplex of this space. In the case of a minimization,

the method consists of replacing the point of the simplex with the highest objective value (the least satisfactory point) by testing several movements (reflection, expansion, contraction, etc.).

For application, this method only requires the value of the criterion at certain points in the state space, with no other information. It is therefore suited to the criteria that are non-derivable or for which the calculation of an approximation of a gradient would be very costly. Moreover, the method is not particularly sensitive to noise around this criterion, making it robust. See [CON 09] and [KAR 07] for more detail on this method.

1.2.3.2. *First-order methods*

To apply this type of local method, we require the gradient (or an approximation of the gradient) of each point of the state space. In the context of a minimization, the principle of these methods consists of moving in an iterative manner in the opposite direction to the gradient (or another descending direction) at the level of the current point. This is equivalent to replace the function f by its local linear model:

$$f(\vec{x} + \vec{h}) = f(\vec{x}) + \nabla_f(\vec{x}).\vec{h} + o(\|\vec{h}\|)$$

Supposing we start from point $\vec{x}_n$, the following point is given by:

$$\vec{x}_{n+1} = \vec{x}_n - \mu \nabla_f(\vec{x}_n)$$

with $\mu > 0$ as the parameter of the algorithm. We therefore verify:

$$f(\vec{x}_{n+1}) = f(\vec{x}_n) - \mu \|\nabla_f(\vec{x}_n)\|^2 + o(\mu \|\nabla_f(\vec{x}_n)\|)$$

which implies the reduction of f for a sufficiently small μ. This method is often modified by allowing μ to vary with each iteration

$$\vec{x}_{n+1} = \vec{x}_n - \mu_n \nabla_f(\vec{x}_n)$$

Variants of this algorithm differ in the choice of direction of descent and in the step size μ_n. These methods converge slowly on functions which are very different to the linear model [BER 99, NOC 06].

1.2.3.3. *Second-order methods*

Here, we suppose that f is of the class C^2 and that we are able to calculate its second derivatives. The principle of this method consists of constructing, locally, a quadratic model $q(\vec{x})$ of function f and seeking the minimum ($\vec{x}^*$) associated with the model. The point $\vec{x}^*$ thus becomes the current point for the following iteration.

In the vicinity of a point, $\vec{x}_k$, we therefore approach f by the quadratic function given by the second-order Taylor formula:

$$\begin{array}{l} q(\vec{x}) = f(\vec{x}_k) \\ +(\vec{x} - \vec{x}_k)^T . \nabla_f(\vec{x}_k) \\ +\frac{1}{2}(\vec{x} - \vec{x}_k)^T . \nabla_f^2(\vec{x}_k) . (\vec{x} - \vec{x}_k). \end{array}$$

We thus obtain the following recursion (in the case where the Hessian matrix $\left[\nabla_f^2(\vec{x}_k)\right]$ is invertible):

$$\vec{x}_{k+1} = \vec{x}_k - \left[\nabla_f^2(\vec{x}_k)\right]^{-1} . \nabla_f(\vec{x}_k).$$

The Newton method is generally more efficient than the gradient-based approach and converges in one iteration on positively defined quadratic forms. However, the global

convergence[1] is not guaranteed as the Hessian may not be invertible for certain points. The complexity of the algorithm increases following the cube of the dimension of the problems under consideration due to the inversion of the Hessian in the recursion.

In practice, we generally solve the following system: $\left[\nabla_f^2(\vec{x}_k)\right].\vec{d_k} = -\nabla_f(\vec{x}_k)$.

Variable metric (quasi-Newton) methods are a robust alternative to the Newton method. Initially developed by Broyden, Fletcher, Goldfarb and Shanno, the BFGS method (the name uses the initials of the four authors who discovered it, independently, in 1970) [BRO 70, FLE 70, GOL 70, SHA 70], allows us to construct a positively defined approximation of the Hessian matrix at each point of the state space. Global convergence is therefore guaranteed and occurs considerably faster than when using the gradient algorithm. For large-scale problems, a limited memory variation exists: LM-BFGS [NOC 80].

1.2.4. *Local methods subject to constraints*

Let us consider the following problem:

$$\begin{array}{l}\min f(\vec{x}) \\ s.c\ g_i(\vec{x}) \leq 0\end{array}$$

1.2.4.1. *Projection method*

In this case, the direction of descent $\vec{\delta_x}$ is calculated using a method without constraints. If the result is then found to fall outside the domain, we project the point onto the constraints. In practice, we may face the following difficulties:

---

1 Convergence for any initial point.

– calculation of the direction of projection;

– if several constraints are saturated, how should we choose the one on which to project?

– difficulties of projection onto nonlinear constraints.

1.2.4.2. *Linearization of the problem*

In this case, we linearize the criterion and the constraints.

$$\begin{cases} f(\vec{x}+\vec{\delta_x}) \simeq f(\vec{x}) + \left(\vec{\nabla}_{\vec{x}} f\right)^T \vec{\delta_x} \\ g_i(\vec{x}+\vec{\delta_x}) \simeq g_i(\vec{x}) + \left(\vec{\nabla}_{\vec{x}} g_i\right)^T \vec{\delta_x} \leq 0 \end{cases}$$

We then return to the following problem:

Minimization of $\left(\vec{\nabla}_{\vec{x}} f\right)^T \vec{\delta_x}$ in relation to $\vec{\delta_x}$ with constraints:

$$\vec{d_x}_{min} \leq \vec{\delta_x} \leq \vec{d_x}_{max}$$

1.2.4.3. *Penalizations*

The penalization method consists of optimizing a new function which takes into account the constraints:

$$\phi(\vec{x}) = f(\vec{x}) + \lambda(h(\vec{g}(\vec{x}))) \ \lambda > 0, \ h(.) \geq 0$$

Using the external penalization method, h is such that:

$h(\alpha) = 0$ if $\alpha \leq 0$
$h(\alpha)$ increases with α if $\alpha > 0$

$\Rightarrow$ if $\vec{g}(\vec{x}) \leq 0$ minimizing $f(\vec{x})$
$\Rightarrow$ if $\vec{g}(\vec{x}) > 0,$ the added penalization term increases.

Using this method, it is possible that certain constraints may be slightly overstepped. If we wish to eliminate this possibility, it is better to use the interior penalization (see [BOU 68]).

1.2.5. *Deterministic global methods*

1.2.5.1. *Enumeration*

Enumeration offers an interesting alternative for discrete problems in state spaces of low dimension. This method consists of evaluating each point in the state space and identifying the point with the highest (or the lowest) criterion value. This is the only possible approach in cases where the locality principle is not respected (this principle indicates that it is possible to approximate (or calculate) a slope at each point in the state space). The use of this kind of algorithm is interesting when the number of points for evaluation is relatively small. In practice, however, many search spaces are too large for us to find all possible solutions.

1.2.5.2. *Branch and Bound*

This method was initially proposed by A.H. Land and A.G. Doigien in 1960 [LAN 60] in the context of solving linear problems using integers.

To apply this method, we need a lower bound (in the case of a minimization) of the criterion for any subspace of the state space $\mathcal{X}$ and a principle for dividing a subspace $\mathcal{X}_i$ into $K > 2$ subspaces $\mathcal{X}_{i1}, ..., \mathcal{X}_{iK}$.

If an optimal solution is found in subspace $\mathcal{X}_i$, it is not necessarily optimal for $\mathcal{X}$ as a better solution may be found later in the process when evaluating unexplored areas.

If the lower bound of a given subspace is greater than the best optimal solution found, the global solution will not be

found in this subspace, and therefore exploration of this subspace may be stopped.

The Branch and Bound (B&B) principle is therefore as follows:

– Divide a problem into subproblems. We partition $\mathcal{X}$ into a finite collection of subsets $\mathcal{X}_1$, $\mathcal{X}_2$, ..., $\mathcal{X}_K$.

– Use bounds for the optimal cost in order to avoid exploring certain parts of the set of admissible solutions.

– The subproblems may be as difficult as the original problem. In such cases, the subproblems themselves are divided.

The success of this method largely depends on the precision of the bound associated with a subspace $\mathcal{X}_i$. Thus, the closer the lower bound $b(\mathcal{X}_i)$ is to the true minimum of $\mathcal{X}_i$, the more effective the method will be. The main field of application of this method is integer LP, which uses a bound based on the relaxation of the problem in real numbers computed using classic LP.

Interval arithmetic may be used to produce effective bounds for continuous state spaces. This particular form of B&B is known as "interval programming" [KEA 01].

B&B allows us to treat large-scale problems on the condition that we have a reliable boundary. In other cases, it is better to look to stochastic approaches. See Jaulin *et al.* [JAU 88] for more information on B&B method.

1.2.5.3. *The "Tunneling" method*

Alfufi-Pentini *et al.* and Levy and Montalvo [ALF 85, LEV 85] describe the foundations of this method, and Cetin *et al.* [CET 93] discuss the most advanced concepts in this domain.

Principle: the two following steps are carried out in an iterative manner:

1) Seek a local optimum $\vec{x}_{loc}$.

2) Eliminate $\vec{x}_{loc}$. To do this, we construct a tunneling function T *maximal* in $\vec{x}_{loc}$ and we carry out a local search on the new function T.

Drawbacks of the method:

– The tunneling function is difficult to create.

– Local minimization of the tunneling function is difficult.

– The method does not always produce the global optimum.

1.2.5.4. *Covering methods*

Principle: If f is *Lipschitz* with constant L, then:

$$\forall(\vec{x},\vec{z}) \in \mathbb{R}^n \times \mathbb{R}^n |f(\vec{x}) - f(\vec{z})| \leq L||\vec{x} - \vec{z}||$$

thus, if we know the value of f at N points $\vec{x}_1, \vec{x}_2, ..., \vec{x}_N$ of $\mathcal{X}$ (search space), we can determine sets $\mathcal{X}_i$ $(i = 1...N)$ such that:

$$\mathcal{X}_i \subset \{x \in \mathcal{X} | f(\vec{x}) \geq f(\vec{x}_i) - \delta\}$$

where $\delta > 0$ is fixed. Thus, if the points $\vec{x}_1, \vec{x}_2, ..., \vec{x}_N$ are chosen so that $(\mathcal{X}_1, \mathcal{X}_2, ..., \mathcal{X}_N)$ covers $\mathcal{X}$, so:

$$\mathcal{X} \subset \left(\cup_{i=1}^{N} \mathcal{X}_i\right)$$

then the global optimum $\vec{x}_{opt}$ is known with precision δ, that is:

$$\left(\min_{1 \leq i \leq N} f(\vec{x}_i)\right) - \delta \leq f(\vec{x}_{opt})$$

An important specific case:

$$\mathcal{X}_i = B(\vec{x}_i, \varepsilon) = \{\vec{x} \in \mathcal{X} / \|\vec{x} - \vec{x}_i\| \leq \varepsilon\} \; i = 1...N$$

($B(\vec{x}_i, \varepsilon)$ closed ball of center $\vec{x}_i$ and radius ε) then the problem is solved with precision $\delta = L\varepsilon$. Constant L, *a priori* unknown, is estimated by the algorithm. If we know the value of f at k points $\vec{x}_1, \vec{x}_2, ..., \vec{x}_k$, we may use:

$$L_k = \max_{1 \leq j \leq k} \frac{|f(\vec{x}_i) - f(\vec{x}_j)|}{\|\vec{x}_i - \vec{x}_j\|}$$

Advantages of the method:

– simple to implement;

– theoretically interesting;

– parallelizable method.

Drawback of the method:

– inefficient for large-scale problems.

1.2.5.5. *Continuous deformation methods*

The principle of continuous deformation methods consists of gradually deforming a function for optimization, f_0, toward the desired objective function f_1:

$$\begin{array}{ccc} & H(t, \vec{x}),\ t \in [0,1] & \\ \min_{\vec{x} \in E} f_0(\vec{x}) & \rightarrow & \min_{\vec{x} \in E} f_1(\vec{x}) \\ f_0(\vec{x}) = H(0, \vec{x}) & & f_1(\vec{x}) = H(1, \vec{x}) \end{array}$$

Function H in the schema above is, mathematically speaking, a *homotopy*, i.e. a continuous function in the domain $[0,1] \times E$. Intuitively, f_0 is continuously deformed into f_1 when t varies from 0 to 1. In practice, we ensure that f_0 only possesses a single optimum, which may be sought using

an efficient local method. The difficulty therefore consists of finding a path $(t, \gamma(t))$ of $[0,1] \times E$ so that the following property verifies:

$$\forall t \in [0,1]\,,\, \forall \vec{x} \in E\,,\, H(t,\gamma(t)) \leq H(t,\vec{x})$$

Intuitively, we know that all points of the path $(t, \gamma(t))$ are optimal for E. The starting point $(0, \gamma(0))$ is obtained by local optimization of f_0. The path $(t, \gamma(t))$ is constructed step by step.

It is not possible to construct the path $(t, \gamma(t))$ without additional hypotheses: we will presume that the homotopy H is such that $\frac{\partial H}{\partial x}$ is continually differentiable for $[0,1] \times E$. Using this hypothesis and a condition stating that the path must start at a minimum of f_0, we obtain:

$$\frac{\partial^2 H}{\partial t \partial x}(t,\gamma(t)) + \dot{\gamma}(t)\frac{\partial^2 H}{\partial x^2}(t,\gamma(t)) = 0.$$

The differential equation above allows us to construct the optimal path. The classic algorithms for solving ordinary differential equations may be used to obtain a numerical solution to the problem.

These methods are efficient for continuous state spaces and are most effective for molecular conformation problems in the pharmaceutical industry. For more information on these methods, see [DUN 05].

1.2.6. *Stochastic global methods*

1.2.6.1. *Tabu search*

Tabu search is a metaheuristic initially developed by Glover [GLO 86] and, independently, by Hansen [HAN 86], under the name *steepest ascent mildest descent*. This method

is based on simple principles but is nevertheless extremely effective, combining a local search procedure with a certain number of mechanisms which prevent it from becoming blocked at local optima or from returning to zones which have already been explored [CHA 96, REE 95]. It has been successfully applied to a number of difficult combinatory optimization problems, including vehicle routing problems [GEN 94], quadratic affectation problems [SKO 90], sequencing problems [WID 89], graph coloration problems [HER 87], etc.

Basic principle: during the first phase, the tabu search method may be seen as a generalization of local improvement methods. Starting with any given solution $\vec{x}$ belonging to the set of solutions $\mathcal{X}$, we move towards a solution $s(\vec{x})$ located in the vicinity $\mathcal{V}(\vec{x})$ of $\vec{x}$. To choose the best neighbor $s(\vec{x})$ in $\mathcal{V}(\vec{x})$, the algorithm evaluates the objective function f at each point of $\mathcal{V}(\vec{x})$, and retains the neighbor which improves the value of the objective function f, or, in the worst cases, which degrades it the least.

The originality of the tabu method when compared to local methods, which stop when no more neighbors $s(\vec{x})$ exist which improve the value of the objective function f, lies in the fact that we retain the best neighbor solution even if this result is worse than the initial solution. This criterion authorizing degradation of the objective function, prevents the algorithm from becoming blocked at local minima, but it does introduce a risk of cycling. Effectively, when the algorithm has left any given minimum by accepting the degradation of the objective function, it may turn back on itself at the following iteration.

To solve this problem, the algorithm requires a memory in order to temporarily retain the trace of the last best solutions found. These solutions are declared *tabu*; hence, the name of the method. They are stored in a list of given length L known as the *tabu list* $\mathcal{T}$. A new solution will only be accepted if it does not feature on the tabu list. This criterion for the

acceptance of new solutions prevents the algorithm from cycling during visits to a number of solutions, which is at least equal to the length of the tabu list, and directs exploration of the method toward unexplored regions of the state space.

The tabu list is generally generated as a *circular* list: at each iteration, we eliminate the oldest tabu solution, replacing it with the new retained solution. However, the coding involved in a list of this type is bulky as we must retain all the elements which define a solution. To counteract this constraint, the tabu list of forbidden solutions is replaced by a list of *forbidden transformations* preventing all transformations which are the inverse of a recent transformation. We thus obtain the following pseudocode:

1) calculate an initial configuration $\vec{x}$

2) $\vec{x}_{best} \leftarrow \vec{x}$

3) $f_{min} \leftarrow f(\vec{x}_{best})$

4) $\mathcal{T} \leftarrow \phi$

5) $k \leftarrow 0$

6) as long as $k < k_{max}$ then

i) $k \leftarrow k + 1$

ii) $\mathcal{C} \leftarrow \mathcal{V}(\vec{x}) - \{m(\vec{x}); \forall m \in \mathcal{T}\}$ (set of candidate configurations)

iii) determine the element $\vec{y} = m_{\vec{x}\vec{y}}(\vec{x})$, which minimizes f for $\mathcal{C}$ ($\vec{y}$ is one of the non-prohibited neighbors of $\vec{x}$);

iv) if $f(\vec{y}) \geq f(\vec{x})$ then add $m_{\vec{x}\vec{y}}^{-1}$ to list $\mathcal{T}$ (eliminating the oldest element of $\mathcal{T}$ if necessary)

v) if $f(\vec{y}) < f(\vec{x})$ then
– $\vec{x}_{best} \leftarrow \vec{y}$
– $f_{min} \leftarrow f(\vec{y})$

7) return $\vec{x}_{best}$

This basic version can be improved by adding an aspiration criterion (temporary removal of the ban on accepting an elementary transformation), intensification principles (deepening of the search in certain regions of the domain) and diversification principles (which encourage exploration) [GLO 88, GLO 91, GLO 92].

In practice, we use a stochastic implementation of this method, which consists of using a stochastic neighborhood of the current solution (we randomly select a number $N < |\mathcal{V}(\vec{x})|$ of configurations in the vicinity of the current solution $\vec{x}$).

The tabu search is thus a simple method, which is easily adapted to all types of problems (discrete or continuous). It requires a list of past transformations, the management of which is the critical point of the method [GLO 88, GLO 91, GLO 92].

1.2.6.2. *Simulated annealing*

Simulated annealing [CER 85, KIR 83] originated in the domain of thermodynamics. This method arose from an analogy with the physical phenomenon of slow cooling found in metals in a state of fusion which leads to a solid, low-energy state. The temperature must be reduced slowly with steps which are sufficiently long for thermodynamic equilibrium to be reached at each temperature increment. For materials, this low energy results in a regular, crystal-like atomic structure.

The annealing process thus consists of bringing a solid into a low energy state after raising its temperature, a process which may be summarized in the following two steps:

– raise the solid to a very high temperature in order to reach the point of fusion;

– cool the solid, following a specific temperature reduction plan in order to reach a solid state with minimal energy.

During the liquid phase, particles are distributed in a random manner. The state of minimal energy is attained if the initial temperature is sufficiently high and the cooling time sufficiently long; if either of these conditions is not respected, the solid enters a metastable state of non-minimal energy (the opposite process to annealing is tempering, which consists of cooling a solid extremely quickly).

Algorithm

In 1953, Metropolis [MET 53] developed an algorithm to simulate the physical process of annealing on a computer. Given a current state i of energy E_i, a state is generated by applying a disturbance which transforms the current state into a new state.

– If $E_j - E_i \leq 0$, the state j is accepted as the new current state.

– If $E_j - E_i > 0$, the state j is accepted as the new current state with probability P_a:

$$P_a = e^{\left(\frac{E_i - E_j}{k_b T}\right)}$$

where T is the temperature and k_b is Boltzmann's constant.

The temperature influences the probability of acceptance of a higher energy state. For a high temperature, the probability of acceptance at any given movement tends toward 1: all changes will be accepted. If the cooling process is sufficiently slow, the solid assumes the state of equilibrium at each temperature increment. In the Metropolis algorithm, this equilibrium is reached by generating a large number of transitions at each temperature. Thermic equilibrium is characterized by the Boltzmann statistical distribution. This

distribution gives the probability that the solid will be in a state i of energy E_i at temperature T:

$$P_r\{X = i\} = \frac{1}{Z(T)} e^{-\left(\frac{E_i}{k_b T}\right)}$$

where X is the random variable associated with the current state of the solid and $Z(T)$ is the distribution function of X allowing normalization:

$$Z(T) = \sum_{j \in S} e^{-\left(\frac{E_j}{k_b T}\right)}$$

In the simulated annealing algorithm, we can apply the Metropolis algorithm to generate a sequence of solutions in the state space S. To do this, we create an analogy between a multi-particular system and our optimization problem using the following equivalences:

– The admissible solutions represent the possible states of the solid.

– The function to optimize represents the energy of the solid.

We then introduce a control parameter C, which plays the role of the temperature.

Let C_k be the value of this parameter and L_k the number of transitions generated at iteration k. Using this notation, we can summarize the principle of simulated annealing in the following manner.

At the beginning of the process, the values of C_k are high, allowing us to accept transitions with major degradation of the criterion, thus exploring the state space in a homogeneous manner. As C_k decreases, only transitions which improve or

barely damage the criterion are accepted. Finally, as C_k tends toward zero, no deterioration of the criterion will be accepted, and the simulated annealing algorithm behaves in the same way as a local search algorithm.

Algorithm 1.1. Simulated annealing

Require: $\vec{x}_i, C_0, L_0, k = 0$
  repeat
    for $l = 0 \rightarrow L_k$ **do**
      Generate a solution $\vec{x}_j$ from the neighborhood $S_{\vec{x}_i}$ of the current solution $\vec{x}_i$;
      If $f(\vec{x}_j) < f(\vec{x}_i)$, then $\vec{x}_j$ becomes the current solution;
      Otherwise, $\vec{x}_j$ becomes the current solution with probability $p = e^{\left(\frac{f(\vec{x}_i) - f(\vec{x}_j)}{C_k}\right)}$
    end for
    k=k+1
    Calculate (L_k, C_k)
  until $C_k \simeq 0$

The simulated annealing algorithm may be used to solve a large number of combinatory optimization problems with properties of stochastic convergence to an optimal solution, but presents the drawback of only working on a single point of the state space (something which is problematic in cases with several quasi-optimal solutions) and is not suitable for multi-objective optimization. See [AAR 89, ING 89, ING 96] for additional information on this technique.

1.2.6.3. *Stochastic Branch and Bound*

Principle: the principle of Stochastic Branch and Bound method is the same as the deterministic branch and bound method, with the use of statistical results in order to eliminate parts of the search area which do not contain the global optimum.

Let us take the following problem:

$$\begin{cases} \min f(\vec{x}) \\ \vec{x} \in \mathcal{X} \end{cases}$$

Let η be the random variable for which realizations are the values of f and of which the distribution function is F. We wish to find an estimator of the theoretical minimum M of the random variable η defined by:

$$\begin{cases} P(\eta \geq M) = 1 \text{ and} \\ \forall \varepsilon > 0 \; P(\eta \geq M + \varepsilon) < 1 \end{cases}$$

Under certain hypotheses on the function F, we can determine an optimal linear estimator of M, denoted M_N, using the N values of f in the sample $\kappa = \{\vec{x}_1, \vec{x}_2, ..., \vec{x}_N\}$.

We are also able to calculate the confidence interval of asymptotic level $1 - \gamma$ associated with M of the form:

$$l_{N,\gamma} = [\inf_{N,\gamma}(f), \sup_{N,\gamma}(f)]$$

where $\gamma > 0$ is fixed.

We thus obtain:

$$\lim_{N \to \infty} P[M \in l_{N,\gamma}] = 1 - \gamma$$

Advantages of the method:

– Robustness.

– The parameters N, γ, etc. are easily controlled by the user.

– Tests have shown a linear evolution of the number of points generated with the size of the problem.

This is widely recognized as one of the most effective methods.

Drawback of the method: there are currently no demonstrations of stochastic convergence associated with this method.

1.2.7. *Genetic algorithms*

Genetic algorithms are inspired by the theory of evolution proposed by Charles Darwin in the 19th Century.

According to Darwin's theory, a population of individuals evolves through the mechanisms of sexual reproduction. Those individuals who are best suited to their environment reproduce more than other individuals, thus promoting the most appropriate characteristics. For example, a giraffe with a longer neck than others of its species will have access to more food, and consequently has an improved chance of survival and reproduction. The descendants of this giraffe will also have particularly long neck, and the average neck length in the giraffe population will increase. An algorithm based on this theory was first proposed by John Holland in the early 1970s [HOL 75]. From a set of approximate solutions (the population), we select two good solutions and recombine them to produce a new solution. At the same time, we generate new genes using a mutation operator in order to promote exploration of the state space. In parallel with this process, we eliminate the least suitable solutions using a selection process. By repeating this process, the adaptation of the population increases and converges to a solution to the problem.

These algorithms will be discussed in greater detail in Chapter 2.

1.2.8. *Conclusion*

As we have seen, a wide variety of optimization algorithms exist, each suited to a certain category of problems. These algorithms may be classified into the following:

– Deterministic:

- local: LP and NLP;

- global: enumeration, B&B, continuous deformation methods.

– Stochastic:

- Global: tabu search, simulated annealing, evolutionary algorithms, stochastic branch and bound.

The associated performances are shown in table represented in Figure 1.3.

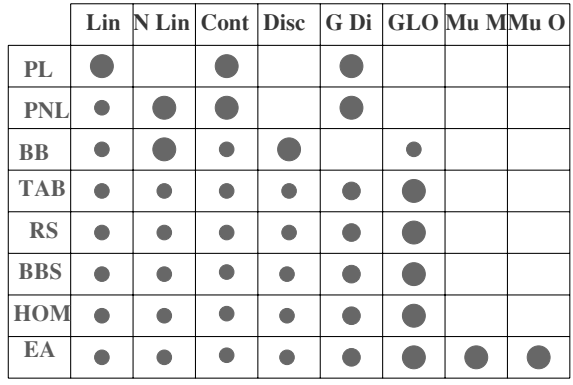

Figure 1.3. *Comparison of performances of optimization. When a method can be used for a class of problems, a dot is shown where the line and column meet. The size of the point reflects how suitable the method is for the class of problems: the larger the point, the better suitable the method is for the specified problem type*

The top row of the figure represents the classes of problems:

– Lin: linear problem;

– N Lin: nonlinear problem;

– Cont: continuous state space;

– Disc: discrete state space;

– L Di: large dimension;

– GLO: global optimum search;

– Mu M: multi-mode search (problem with several quasi-equivalent optima);

– Mu O: multi-objective problem.

The column lists the main classes of optimization algorithms:

– LP: linear programming;

– NLP: nonlinear programming;

– BB: branch and bound;

– TAB: tabu method;

– SA: simulated annealing;

– SBB: sotchastic branch and bound;

– HOM: homotopic approach;

– AE: artificial evolution.