Open access • Book Chapter • DOI:10.1007/978-3-319-91086-4_1

Simulated annealing: From basics to applications — Source link

Daniel Delahaye, Supatcha Chaimatanan, Marcel Mongeau

Institutions: École nationale de l'aviation civile, Geo-Informatics and Space Technology Development Agency

Published on: 01 Jan 2019 - Operations Research and Management Science (Springer)

Topics: Simulated annealing, Metaheuristic, Knapsack problem, Travelling salesman problem and Population

Simulated annealing: From basics to applications

Daniel Delahaye, Supatcha Chaimatanan, Marcel Mongeau

Daniel Delahaye ENAC, Toulouse, France, daniel.delahaye@enac.fr Supatcha Chaimatanan Geo-I

Marcel Mongeau ENAC, Toulouse, France, marcel.mongeau@enac.fr

Simulated Annealing: From Basics to Applications

Daniel Delahaye, Supatcha Chaimatanan and Marcel Mongeau

October 8, 2018

Abstract

Simulated Annealing (SA) is one of the simplest and best-known meta-heuristic method for addressing the difficult *black box* global optimization problems (those whose objective function is not explicitly given and can only be evaluated via some costly computer simulation). It is massively used on real-life applications. The main ad- vantage of SA is its simplicity. SA is based on an analogy with the physical annealing of materials that avoids the drawback of the Monte-Carlo approach (which can be trapped in local minima), thanks to an efficient Metropolis acceptance criterion. When the evaluation of the objective-function results from complex simulation processes manipulating a large-dimension state space involving much memory, population-based algorithms are not applicable and SA is the right answer to address such issues. This chapter is an introduction to the subject. It presents the principles of local search optimization algorithms, of which simulated annealing is an extension, and the Metropolis algorithm, a basic component of SA. The basic SA algorithm for optimization is described together with two theoretical aspects that are fundamental to SA: statistical equilibrium (inspired from elementary statistical physics) and asymptotic convergence (based on Markov chain theory). The chapter surveys the following practical issues of interest to the user that wishes to implement the SA algorithm for its particular application: finite-time approximation of the theoretical SA, polynomial-time cooling, Markov-chain length, stopping criteria, and simulated-based evaluations. To illustrate these concepts, this chapter presents the straightforward application of SA to two classical and simple

classical NP-hard combinatorial optimization problems: the knapsack problem and the traveling salesman problem. The overall SA methodology is then deployed in detail on a real-life application: a large-scale aircraft trajectory planning problem involving nearly 30,000 flights at the European continental scale. This exemplifies how to tackle nowadays complex problems using the simple scheme of SA by exploiting particular features of the problem, by integrating astute computer implementation within the algorithm, and by setting user-defined parameters empirically, inspired by the SA basic theory presented in this chapter.

1 Introduction

Simulated Annealing (SA) is one of the simplest and best-known metaheuristic methods for addressing the difficult *black box* global optimization problems (those whose objective function is not explicitly given and can only be evaluated via some costly computer simulation). It is massively used in real-life applications. The expression "simulated annealing" yields over one million hits when searching through the Google Scholar web search engine dedicated to the scholarly literature.

This chapter is an introduction to the subject. It is organized as follows. The first section introduces the reader to the basics of the simulated annealing algorithm. Section 2 deals with two fundamental theoretical aspects of SA: statistical equilibrium and asymptotic convergence. Practical issues of interest when implementing SA are discussed in Section 3: finite-time approximation, polynomial-time cooling, Markov-chain length, stopping criteria and simulation-based evaluations. Section 4 illustrates the application of SA to two classical NP-hard combinatorial optimization problems: the knapsack problem and the traveling salesman problem. A real-life application, large-scale aircraft trajectory planning problem, is finally tackled in Section 5 in order to illustrate how the particular knowledge of an application and astute computer implementation must be integrated within SA in order to tackle nowadays complex problems using the simple scheme of SA.

2 Basics

In the early 1980s three IBM researchers, Kirkpatrick, Gelatt and Vecchi [12], introduced the concepts of annealing in combinatorial optimization. These concepts are based on a strong analogy with the physical annealing of materials. This process involves bringing a solid to a low energy state after raising its temperature. It can be summarized by the following two steps (see Figure 1):

- Bring the solid to a very high temperature until "melting" of the structure;
- Cooling the solid according to a very particular temperature decreasing scheme in order to reach a solid state of minimum energy.

In the liquid phase, the particles are distributed randomly. It is shown that the minimum-energy state is reached provided that the initial temperature is sufficiently high and the cooling time is sufficiently long. If this is

not the case, the solid will be found in a metastable state with non-minimal energy; this is referred to as *hardening*, which consists in the sudden cooling of a solid.

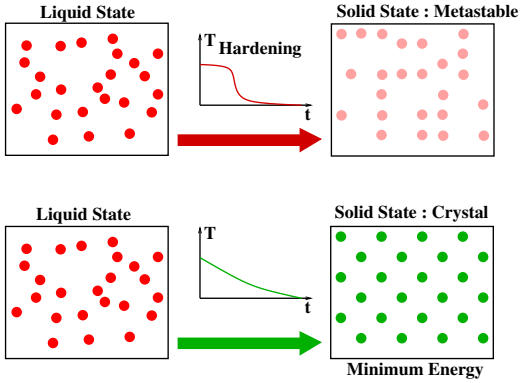

Figure 1: When temperature is high, the material is in a liquid state (left). For a hardening process, the material reaches a solid state with non-minimal energy (metastable state; top right). In this case, the structure of the atoms has no symmetry. During a slow annealing process, the material reaches also a solid state but for which atoms are organized with symmetry (crystal; bottom right).

Before describing the simulated annealing algorithm for optimization, we need to introduce the principles of local search optimization algorithms, of which simulated annealing is an extension.

2.1 Local search (or Monte Carlo) algorithms

These algorithms optimize the cost function by exploring the neighborhood of the current point in the solution space.

In the next definitions we consider (S, f) an instantiation of a combinatorial optimization problem (S: set of feasible solutions, f: objective function to be minimized).

Definition 1 *Let $\mathcal{N}$ be an application that defines for each solution $i \in S$ a subset $S_i \subset S$ of solutions "close" (to be defined by the user according to the problem of interest) to the solution i. The subset S_i is called the neighborhood of solution i.*

In the next definitions, we consider that $\mathcal{N}$ is a neighborhood structure associated to (S, f).

Definition 2 *A generating mechanism is a mean for selecting a solution j in any neighborhood S_i of a given solution i.*

A local search algorithm is an iterative algorithm that begins its search from a feasible point, randomly drawn in the state space. A generation mechanism is then successively applied in order to find a better solution (in terms of the objective function value), by exploring the neighborhood of the current solution. If such a solution is found, it becomes the current solution. The algorithm ends when no improvement can be found, and the current solution is considered as the approximate solution of the optimization problem. One can summarize the algorithm by the following pseudo-code for a minimization problem:

Local search

1. **Draw an initial solution i;**
2. **Generate a solution j from the neighborhood S_i of the current solution i;**
3. **If $f(j) < f(i)$ then j becomes the current solution;**
4. **If $f(j) \geq f(i)$ for all $j \in S_i$ then END;**
5. **Go to step 2;**

Definition 3 *A solution $i^* \in S$ is called a local optimum with respect to $\mathcal{N}$ for (S, f) if $f(i^*) \leq f(j)$ for all $j \in S_{i^*}$.*

Definition 4 *The neighborhood structure $\mathcal{N}$ is said to be exact if, for every local optimum with respect to $\mathcal{N}$, $i^* \in S$, i^* is also a global optimum of (S, f).*

Thus, by definition, local search algorithms converge to local optima unless one has an exact neighborhood structure. This notion of exact neighborhood is theoretical because it generally leads in practice to resort to a complete enumeration of the search space.

Intuitively, if the current solution "falls" in a subdomain over which the objective function is convex, the algorithm remains trapped in this subdomain, unless the neighborhood structure associated with the generation mechanism can reach points outside this subdomain.

In order to avoid being trapped in local minima, it is then necessary to define a process likely to accept current state transitions that momentarily

reduce the performance (in terms of objective) of the current solution: this is the main principle of simulated annealing function such that.

Before describing this algorithm, it is necessary to introduce the Metropolis algorithm [15] which is a basic component of SA.

2.2 Metropolis Algorithm

In 1953, three American researchers (Metropolis, Rosenbluth, and Teller [15]) developed an algorithm to simulate the physical annealing, as described in Section 2. Their aim was to reproduce faithfully the evolution of the physical structure of a material undergoing annealing. This algorithm is based on Monte Carlo techniques which consist in generating a sequence of states of the solid in the following way.

Starting from an initial state i of energy E_i, a new state j of energy E_j is generated by modifying the position of one particle.

If the energy difference, $E_i - E_j$, is positive (the new state features lower energy), the state j becomes the new current state. If the energy difference is less than or equal to zero, then the probability that the state j becomes the current state is given by:

$$Pr\{\text{Current state} = j\} = e^{\left(\frac{E_i - E_j}{k_b.T}\right)}$$

where T represents the temperature of the solid and k_B is the Boltzmann constant ($k_B = 1.38 \times 10^{-23}$ joule/Kelvin).

The acceptance criterion of the new state is called the *Metropolis criterion*. If the cooling is carried out sufficiently slowly, the solid reaches a state of equilibrium at each given temperature T. In the Metropolis algorithm, this equilibrium is achieved by generating a large number of transitions at each temperature. The thermal equilibrium is characterized by the *Boltzmann statistical distribution*. This distribution gives the probability that the solid is in the state i of energy E_i at the temperature T:

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

where X is a random variable associated with the current state of the solid and $Z(T)$ is a normalization coefficient, defined as:

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

2.3 Simulated annealing (SA) algorithm

In the SA algorithm, the Metropolis algorithm is applied to generate a sequence of solutions in the state space S. To do this, an analogy is made between a multi-particle system and our optimization problem by using the following equivalences:

- The state-space points represent the possible states of the solid;
- The function to be minimized represents the energy of the solid.

A control parameter c, acting as a temperature, is then introduced. This parameter is expressed with the same units as the objective that is optimized.

It is also assumed that the user provides for each point of the state space, a neighborhood and a mechanism for generating a solution in this neighborhood. We then define the acceptance principle:

Definition 5 *Let (S, f) be an instantiation of a combinatorial minimization problem, and i, j two points of the state space. The* acceptance criterion *for accepting solution j from the current solution i is given by the following probability*:

$$Pr\{\ accept\ j\} = \begin{cases} 1 & if\ f(j) < f(i) \\ e^{\left(\frac{f(i)-f(j)}{c}\right)} & else. \end{cases}$$

where c is the control parameter.

By analogy, the principle of generation of a neighbor corresponds to the perturbation mechanism of the Metropolis algorithm, and the principle of acceptance represents the Metropolis criterion.

Definition 6 *A* transition *represents the replacement of the current solution by a neighboring solution. This operation is carried out in two stages: generation and acceptance.*

In the sequel, let c_k be the value of the temperature parameter, and L_k be the number of transitions generated at some iteration k. The principle of SA can be summarized as follows:

Simulated annealing

1. **Initialization** $i := i_{start}$, $k := 0$, $c_k = c_0$, $L_k := L_0$**;**
2. **Repeat**
3. **For** $l = 0$ **to** L_k **do**
   - **Generate a solution** j **from the neighborhood** S_i **of the current solution** i**;**
   - **If** $f(j) < f(i)$ **then** $i := j$ **(**j **becomes the current solution);**
   - **Else,** j **becomes the current solution with probability** $e^{\left(\frac{f(i)-f(j)}{c_k}\right)}$**;**
4. $k := k+1$**;**
5. **Compute**(L_k, c_k)**;**
6. **Until** $c_k \simeq 0$

One of the main features of simulated annealing is its ability to accept transitions that degrade the objective function.

At the beginning of the process, the value of the temperature c_k is high, which makes it possible to accept transitions with high objective degradation, and thereby to explore the state space thoroughly. As c_k decreases, only the transitions improving the objective, or with a low objective deterioration, are accepted. Finally, when c_k tends to zero, no deterioration of the objective is accepted, and the SA algorithm behaves like a Monte Carlo algorithm.

3 Theory

This section addresses two theoretical aspects that are fundamental to SA: statistical equilibrium and asymptotic convergence. More details and proofs of the theorems cited in this section can be found in the books [1, 13].

3.1 Statistical Equilibrium

Based on the *ergodicity hypothesis* that a particle system can be considered as a set having observable statistical properties, a number of useful quantities can be deduced from the equilibrium statistical system: mean energy, energy distribution, entropy. Moreover, if this particle set is stationary, which is the case when the statistical equilibrium is reached, the probability density associated with the states in the equilibrium phase depends on the energy of the system. Indeed, in the equilibrium phase, the probability that the system is in a given state i with an energy E_i is given by the Boltzmann law:

Theorem 1 *After a sufficient number of transitions with fixed c ("control parameter") and using the following probability of acceptance:*

$$P_c\{accept\ j|S_i\} = \begin{cases} 1 & if\ f(j) < f(i) \\ e^{\left(\frac{f(i)-f(j)}{c}\right)} & else, \end{cases}$$

the simulated annealing algorithm will find a given solution $i \in S$ with the probability:

$$P_c\{X = i\} = q_i(c) = \frac{1}{N_0(c)} e^{\left(-\frac{f(i)}{c}\right)},$$

where X is the random variable representing the current state of the annealing algorithm, and $N_0(c)$ is the *normalization coefficient*:

$$N_0(c) = \sum_{j \in S} e^{\left(-\frac{f(j)}{c}\right)}.$$

Definition 7 *Let A and B be two sets such that $B \subset A$. We define the characteristic function of B, noted $\kappa_{(B)}$, to be the function such that for any $B \subset A$:*

$$\kappa_{(B)}(a) = \begin{cases} 1 & if\ a \in B \\ 0 & else. \end{cases}$$

Corolary 1 *For any given solution i, we have:*

$$\lim_{c \to 0^+} P_c\{X = i\} = \lim_{c \to 0^+} q_i(c) = q_i^* = \frac{1}{|S_{opt}|} \kappa_{(S_{opt})}(i),$$

where S_{opt} represents the set of global optima, and κ is the characteristic function.

This result guarantees the asymptotic convergence of the simulated annealing algorithm towards an element of the set of global optima, provided that the stationary distribution $q_i(c)$ is reached at each value of c.For a discrete state space, such distributions are discrete and one can compute the probability to reach one particular point x_i in the state space with an objective value y_i:

$$q_i(c) = \frac{e^{\left(-\frac{y_i^c}{c}\right)}}{\sum_{j \in S} e^{\left(-\frac{y_j^c}{c}\right)}}$$

The expected value of the function f to optimize at equilibrium for any positive value of c is donoted $\langle f \rangle_c$ and the variance is donoted $\langle f^2 \rangle_c$.

At a very high temperature c, the SA algorithm moves randomly in the state space. With each point x_i addressed by this process, is associated an objective value y_i by the mapping $y_i^c = f(x_i)$. If we consider this process for a long period, it is possible to build the distribution of the objective-function values $y_i^c, (i = 1,2,...,N)$ generated by the SA process. This distribution depends on the temperature c and will be noted $q(c)$. For large values of c, this distribution is equal to the objective distribution. Figure 2 gives an example of such a distribution. The figure shows a one-dimensional objective function for which the circles represent the samples of the SA algorithm at some high temperature c_1. The dashed horizontal line shows the mean of this distribution ($< f(c_1) >$), and on the left-hand side the associated distribution is represented by the dashed graph ($q(c_1)$). For a lower temperature c_2, some transitions in the SA process are not accepted, meaning that the associated distribution $q(c_2)$ is shifted to the lower levels (squares and solid graph on the left) with a lower expected value.

Definition 8 *The entropy at equilibrium is*

$$H_c = \sum_{i \in S} q_i(c) ln(q_i(c)).$$

Corolary 2 *One has:*

$$\frac{\partial \langle f \rangle_c}{\partial c} = \frac{\sigma_c^2}{c^2}$$
$$\frac{\partial S_c}{\partial c} = \frac{\sigma_c^2}{c^3}$$

These last two expressions play an important role in statistical mechanics. We also deduce the following expressions:

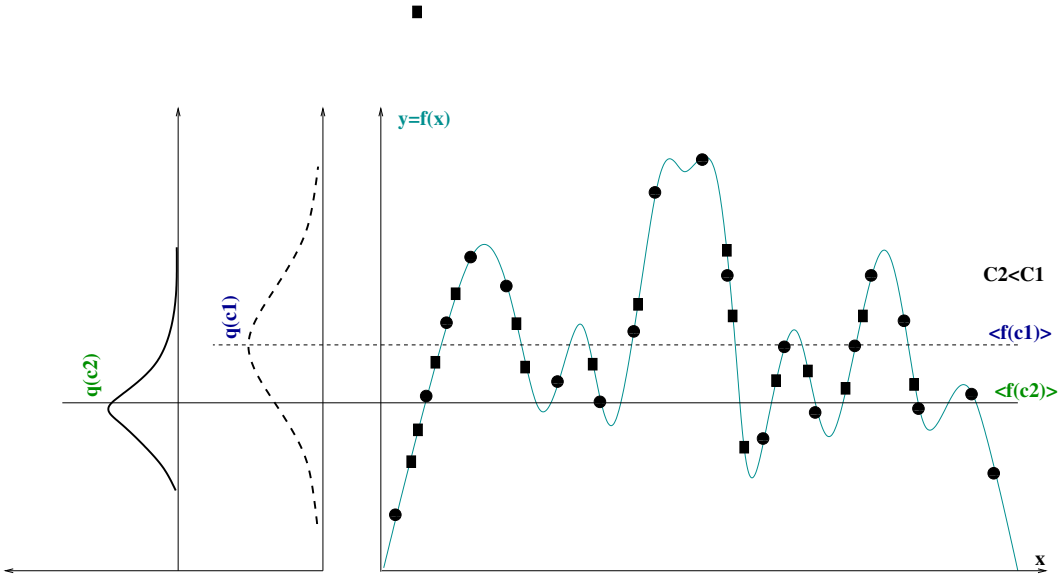

Figure 2: Distribution of the objective-function values at some high temperature c_1 and at a lower temperature c_2.

Corolary 3

$$\lim_{c\to\infty}\langle f\rangle_c = \langle f\rangle_\infty = \frac{1}{|S|}\sum_{i\in S} f(i) \qquad \lim_{c\to 0}\langle f\rangle_c = \langle f\rangle_0 = f_{Opt}$$

$$\lim_{c\to\infty}\sigma_c^2 = \sigma_\infty^2 = \frac{1}{|S|}\sum_{i\in S}(f(i)-\langle f\rangle_\infty)^2 \qquad \lim_{c\to 0}\sigma_c^2 = \sigma_0^2 = 0$$

$$\lim_{c\to\infty} H_c = H_\infty = ln(|S|) \qquad \lim_{c\to 0} H_c = H_0 = ln(|S_{Opt}|)$$

where f_{Opt} denotes the optimal value of f. This last formula represents the third law in thermodynamics (assuming that there is only one state of minimum energy, we then obtain: $S_0 = ln(1) = 0$).

In physics, the entropy measures the level of disorder associated with the system: a high entropy value indicates a chaotic structure, while a low value reflects organization.

In the context of optimization, the entropy is related to a measure of the degree of optimality achieved. During the successive SA iterations, the mathematical expectation of the objective-function value and of the entropy only decrease and converge respectively towards f_{Opt} and $ln(|S_{Opt}|)$.

The derivative of the distribution $q_i(c)$ with the temperature c is given by the following expression:

$$\frac{\partial q_i(c)}{\partial c} = \frac{q_i(c)}{c^2}\left[\langle f\rangle_c - f(i)\right]$$

Since $\langle f\rangle_c \leq \langle f\rangle_\infty$, one can exhibit three regimes in the simulated annealing process. More precisely one can show the following:

Corolary 4 *Let (S, f) be an instantiation of a combinatorial optimization problem with $S_{Opt} \neq S$, and let $q_i(c)$ be the stationary distribution associated with the annealing process. We then have:*

$$(i)\ \forall i \in S_{Opt}\ \frac{\partial q_i(c)}{\partial c} < 0;$$

$$(ii)\ \forall i \notin S_{Opt}\ \textit{such that}\ f(i) \geq\ \langle f\rangle_\infty : \frac{\partial q_i(c)}{\partial c} > 0;$$

$$(iii)\ \forall i \notin S_{Opt}\ \textit{such that}\ f(i) <\ \langle f\rangle_\infty, \exists \tilde{c}_i > 0\ \textit{satisfying:}$$

$$\begin{cases} \frac{\partial q_i(c)}{\partial c} > 0\ \textit{if}\ c < \tilde{c}_i \\ \frac{\partial q_i(c)}{\partial c} = 0\ \textit{if}\ c = \tilde{c}_i \\ \frac{\partial q_i(c)}{\partial c} < 0\ \textit{if}\ c > \tilde{c}_i \end{cases}$$

This corollary indicates that the probability of finding an optimal solution increases monotonically when c decreases. Moreover, for any non-optimal solution, there exists a positive value $\tilde{c}_i$ such that for $c < \tilde{c}_i$, the probability of finding this solution decreases as c decreases.

Definition 9 *The* acceptance rate *associated with the simulated annealing algorithm is defined by:*

$$\chi(c) = \frac{\textit{Number of accepted transitions}}{\textit{Number of proposed transitions}}$$

As a general rule, when c has a high value, all transitions are accepted and $\chi(c)$ is close to 1. Then, when c decreases, $\chi(c)$ decreases slowly until reaching 0, indicating that no transitions are accepted.

By observing the evolution of $\langle f\rangle_c$ and σ_c^2 as a function of c, we note that there exists a critical value called the transition threshold (denoted c_t), that delimits two distinct regions of the distribution at equilibrium. This threshold is the value c_t such that

$$\langle f\rangle_{c_t} \approx \frac{1}{2}(< f_\infty > + f_{opt})$$

and

$$\begin{array}{rl} \sigma_c^2 & \approx \sigma_\infty^2 \quad \text{if } c \geq c_t \\ & < \sigma_\infty^2 \quad \text{if } c < c_t \end{array}$$

For any given value of c, the search space S can therefore be partitioned into two regions:

1. Region R_1: where σ_c^2 remains roughly constant (close to σ_∞^2) when c decreases.
2. Region R_2: where σ_c^2 decreases when c decreases.

When c approaches the value of c_t, the acceptance rate is about 0.5 (i.e., $\chi(c_t) \approx 0.5$).

Furthermore, one can show:

- In R_1, for large values of c, $\langle f \rangle_c$ is linear in c^{-1}, and σ_c^2 is roughly constant.
- In R_2, for small values of c, $\langle f \rangle_c$ is proportional to c, and σ_c^2 is proportional to c^2

One can then propose the following approximation models for $\langle f \rangle_c$ and σ_c^2:

$$\begin{cases} \langle f \rangle_c \cong f_< = f_{min} + N_t \left(\langle f \rangle_\infty - f_{min} - \frac{\sigma_\infty^2}{c} \right) \frac{c}{1-\gamma c} \text{ if } c \leq c_t \\ \\ \langle f \rangle_c \cong f_> = \langle f \rangle_\infty - \frac{\sigma_\infty^2}{c} \text{ if } c > c_t \end{cases}$$

$$\begin{cases} \sigma_c^2 = \sigma_<^2 = N_t^2 \sigma_\infty^2 \left(\frac{c}{1-\gamma c} \right) \text{ if } c \leq c_t \\ \\ \sigma_c^2 = \sigma_>^2 = \sigma_\infty^2 \text{ if } c > c_t \\ \\ \text{with} \\ \\ c_t = \frac{2\sigma_\infty^2}{\langle f \rangle_\infty - fmin} \text{ and } N_t = \frac{1-\gamma c_t}{c_t} \end{cases}$$

where, roughly speaking, γ is the first-order approximation of $\langle f \rangle_c$. Finally, let us introduce the *specific heat*, noted $H(c)$ which is given by the following formula:

$$H(c) = \frac{d\langle f \rangle_c}{dc} = \frac{\langle f \rangle_c^2 - \langle f \rangle_c^2}{k_b c^2}$$

A large value of $H(c)$ indicates that the material starts to become solid: in this case, the decreasing rate of the temperature has to be reduced.

3.2 Asymptotic convergence

The simulated annealing algorithm possesses the property of *stochastic convergence* towards a global optimum provided that it provides an infinitely-long temperature decay diagram with infinitely-small decay steps. This decay scheme is purely theoretical and one will try in practice to get closer to this ideal while remaining within reasonable times of execution.

Definition 10 *A Markov chain is a sequence of states, where the probability of reaching a given state depends only on the previous state. Let $X(k)$ be the state reached at the k^{th} iteration. Then, the probability of transition at the k^{th} iteration for each state pair i, j is given by $P_{ij}(k) = Pr\{X(k) = j | X(k-1) = i\}$. The associated matrix $[P_{ij}(k)]$ is called the transition matrix.*

In the simulated annealing context, a Markov-chain transition corresponds to a move in the state space (generation plus acceptance).

Definition 11 *The* transition probabilities of the SA algorithm *are given by:*

$$\forall i, j \in S \; P_{ij}(k) = P_{ij}(c_k) = \begin{cases} G_{ij}(c_k) A_{ij}(c_k) & \text{if } i \neq j \\ 1 - \sum_{l \neq i} P_{il}(c_k) & \text{if } i = j \end{cases} \quad (1)$$

where $G_{ij}(c_k)$ denotes the probability of generating state j from state i; and $A_{ij}(c_k)$ is the probability of accepting the state j generated from the state i. For all $i, j \in S$, $A_{ij}(c_k)$ is given by:

$$A_{ij}(c_k) = e^{\left(-\frac{(f(j)-f(i))^+}{c_k}\right)}$$

$$\text{with } a^+ = \begin{cases} a \text{ if } a > 0 \\ 0 \text{ else} \end{cases}$$

Theorem 2 *Let the transition probability associated with the SA algorithm defined by (1). Suppose that the following condition is satisfied:*

$$\forall i, j \in S \; \exists p \geq 1, \exists l_0, l_1, ..., l_p \in S,$$

with $l_0 = i, l_p = j$, and $G_{l_k, l_{k+1}} > 0, k = 0, 1, ..., p-1$.

Then, the Markov chain has a stationary distribution, noted $q(c)$, whose components are given by:

$$q_i(c) = \frac{1}{N_0(c)} e^{\left(-\frac{f(i)}{c}\right)}, \ \forall i \in S$$

where $N_0(c)$ is the normalization parameter.

Furthermore,

$$\lim_{c \to 0} q(c) = q^*$$

with $q^* = \frac{1}{|S_{opt}|} \kappa_{(S_{opt})}(i)$ $i \in S$.

Finally,

$$\lim_{c \to 0} \lim_{k \to \infty} Pr\{X_k^c = i\} q(c) = q^*$$

and

$$\lim_{c \to 0} \lim_{k \to \infty} Pr\{X_k^c \in S_{opt}\} = 1,$$

where X_k^c denotes the k^{th} iterate obtained at temperature c. This result indicates the convergence of the simulated annealing algorithm to one of the optimal solutions.

Generalization:

Theorem 3 *Assume that the probabilities of generation and acceptance satisfy the following assumptions:*

$$(G_1) \forall c_k > 0, \ \forall i, j \in S \ \exists p \geq 1, \ \exists \ l_0, l_1, ..., l_p \in S \ : \\ l_0 = i \ l_p = j \ and \ G_{l_k l_{k+1}}(c_k) > 0 \ k = 0, 1, .., p-1;$$

$$(G_2) \ \forall c_k > 0, \ \forall i, j \ \in S \ : \ G_{ij}(c_k) = G_{ji}(c_k);$$

$$(A_1) \ \forall c_k > 0, \ \forall i, j, k \ \in S \ : \begin{cases} A_{ij}(c_k) = 1, \ if \ f(i) \geq f(j) \\ A_{ij}(c_k) \in]0,1[, \ if \ f(i) < f(j) \end{cases}$$

$$(A_2) \ \forall c_k > 0, \ \forall i, j, k \ \in S \ with \ f(i) \leq f(j) \leq f(k), \ A_{ik}(c_k) = A_{ij}(c_k) A_{jk}(c_k)$$

$$(A3) \ \forall i, j \ \in S \ with \ f(i) < f(j), \lim_{c_k \to 0^+} A_{ij}(c_k) = 0$$

Then, at any iteration k there exists a stationary distribution $q(c_k)$ whose components are given by:

$$q_i(c_k) = \frac{A_{i_{Opt} i}(c_k)}{\sum_{j \in S} A_{i_{Opt} j}(c_k)} \ \forall \ i \in S \ and \ i_{Opt} \in S_{Opt}.$$

Moreover, for any $i_{Opt} \in S_{Opt}$, we have:

$$\lim_{c_k \to 0^+} q_i(c_k) = \frac{1}{|S_{Opt}|} \kappa_{(S_{Opt})}(i)$$

In practice, it is very hard to find acceptance distributions, other than exponential distributions, that satisfy A_1, A_2, A_3.

The theoretical results presented above are not directly applicable to a practical SA algorithm since they assume an infinite number of iterations for each value of c_k, which moreover decreases continuously towards zero.

In the case where the number of iterations at each temperature step is finite, the simulated annealing can be modeled using a Markovian inhomogeneous model for which similar results can be established.

The simulated annealing algorithm converges towards an optimal solution of the optimization problem but it reaches this optimum only for an infinite number of transitions. The approximation of the asymptotic behavior requires a number of iterations whose order of magnitude is equal to the cardinality of the state space, which is unrealistic in the context of NP-hard problems. It is therefore necessary to see the annealing as a mechanism for approaching the global solution of a combinatorial optimization problem, to which it will be necessary to add a local search method allowing an optimum to be reached exactly. In other words, the simulated annealing makes it possible to move in the right attraction basin, and a local method completes the optimization process by determining a local optimum within this basin of attraction corresponding to a global optimum of the problem.

4 Practical issues

This section surveys the following practical issues of interest to the user that wishes to implement the SA algorithm for its particular application: finite-time approximation, polynomial-time cooling, Markov-chain length, stopping criteria, and simulated-based evaluations.

4.1 Finite-Time Approximation.

In practice, the convergence conditions will be approximated by choosing, at every iteration k, relatively small steps of decay of the parameter c_k and a sufficiently large number, L_k, of transitions at this temperature. Intuitively, the greater the decrement, the greater the length of the stabilization steps to

achieve a *quasi-equilibrium* (defined below). There is therefore a trade-off to find between "large decrement" and "length" L_k.

A finite-time implementation of a simulated annealing algorithm can be achieved by generating homogeneous Markov chains of finite length for a finite decreasing sequence of values of the control parameter c.

Definition 12 *A cooling process is defined by:*

1. *A finite sequence of values of the control parameter c, that is to say:*
   - *An initial value c_0;*
   - *A decay function of parameter c;*
   - *A final value for c.*
2. *A finite number of transitions for each value of the control parameter, i.e. a finite length of the associated Markov chain.*

Definition 13 *Let ε be a sufficiently small positive value, k a given iteration number, L_k the length of the k^{th} Markov chain and c_k the value of the control parameter. We say that we have a quasi-equilibrium if the probability distribution of the solutions after L_k iterations of the Markov chain (distribution denoted by $a(L_k, c_k)$) is sufficiently close to the stationary distribution $q(c_k)$:*

$$q_i(c_k) = \frac{1}{N_0(c_k)} e^{-\frac{f(i)}{c_k}} \ \forall\, i \in S$$

$$N_0(c_k) = \sum_{j \in S} e^{-\frac{f(j)}{c_k}}$$

That is :

$$||a(L_k, c_k) - q(c_k)|| < \varepsilon$$

The cooling process using the quasi-equilibrium principle is based on the following observation. When the parameter c_k tends to ∞, the stationary distribution is given by a uniform law on the set of possible solutions S:

$$\lim_{c_k \to \infty} q(c_k) = \frac{1}{|S|} \vec{1},$$

where $\vec{1}$ is the vector of dimension $|S|$ whose components are all one.

Thus, for c_k sufficiently large, each point of the search space is visited with the same probability and a state of quasi-equilibrium is directly reached whatever the value of L_k. Then, the cooling process consists in determining the value (L_k, c_k) that will lead to a quasi-equilibrium at the end of each Markov chain.

There are many possible cooling processes but the two most common ones are the *geometric process* proposed by Kirkpatrick [11, 12] and the *polynomial-time cooling* proposed by Aart and Van Laarhoven [2, 3].

4.2 Geometric cooling.

- **Initial Temperature** This prior heating is performed so that we can find a value of c_0 large enough so that nearly all transitions are accepted at the first iterations. In order to find such a value, one starts with a small value c_0. Then; thie value is progressively multiplied by a number greater than 1 until the acceptance rate $\chi(c_0)$ is close to 1.
- **Decay of the control parameter**: $c_{k+1} := \alpha c_k$ where typically $0.8 < \alpha < 0.99$
- **Stopping criterion**: One decides that the algorithm is terminated when the current solution does not change any longer from one iteration to the next during a sufficiently large number of iterations.
- **Length of the chain**: In theory, it is necessary to allow each chain to reach a state of quasi-equilibrium. To this end, a sufficient number of acceptable transitions must be performed, which generally depends on the problem. Since the number of accepted transitions decrease over time with respect to the number of proposed transitions L_k, the latter must be lower bounded.

4.3 Cooling in polynomial time

Let us explain how the initial value of the temperature parameter can be set and how it should then be iteratively decreased.

4.3.1 Initial temperature c_0

Let m_1 be the total number of transitions proposed that improves strictly the value of objective function, and let m_2 be the number of other (indreasing) proposed transitions. Moreover, let $\bar{\Delta_f}^{(+)}$ be the average of the cost differences over all the increasing transitions. Then, the acceptance rate can be approximated by:

$$\chi(c) \simeq \frac{m_1 + m_2 e^{-\left(\frac{\bar{\Delta_f}^{(+)}}{c}\right)}}{m_1 + m_2}$$

which yields

$$c \simeq \frac{\bar{\Delta_f}^{(+)}}{ln\left(\frac{m_2}{m_2.\chi(c)-m_1.(1-\chi(c))}\right)} \tag{2}$$

The proposed initial value of c_0 is then defined as follows:

Initially c_0 is set to zero. Thereafter, a sequence of m_0 transitions is generated for which the values of m_1 and m_2 are computed. The initial value of c_0 is then calculated from equation (2), where the value of the acceptance rate, $\chi(c)$, is defined by the user. The final value of c_0 is then taken as the initial value in the cooling process.

4.3.2 Decay of the control parameter

The quasi-equilibrium condition is replaced by:

$$\forall k \geq 0 \; ||q(k) - q(k+1)|| < \varepsilon,$$

Thus, for two successive values c_k and c_{k+1} of the control parameter, it is desired for the stationary distributions to be close. This can be quantified by the following formula:

$$\forall i \in S \; \frac{1}{1+\delta} < \frac{q_i(c_k)}{q_i(c_{k+1})} < 1+\delta, \tag{3}$$

where δ is some small positive number *a priori* given. The following theorem provides a necessary condition for satisfying equation (3).

Theorem 4 *Let $q(c_k)$ be the stationary distribution of the Markov chain associated with the simulated annealing process at iteration k, and let c_k and c_{k+1} be two successive values of the control parameter with $c_{k+1} < c_k$, then (3) is satisfied if:*

$$\forall i \in S \quad e^{\Delta_i\left(\frac{1}{c_{k+1}}-\frac{1}{c_k}\right)} < 1+\delta, \tag{4}$$

where $\Delta_i = f(i) - f_{Opt}$.

The necessary condition (4) can be rewritten as:

$$\forall i \in S \quad c_{k+1} > \frac{c_k}{1+\frac{c_k.ln(1+\delta)}{f(i)-f_{opt}}} \tag{5}$$

One can show that the latter condition (5) can be approximated by:

$$\forall i \in S \quad c_{k+1} > \frac{c_k}{1 + \frac{c_k ln(1+\delta)}{3\sigma_{c_k}}} \tag{6}$$

where σ_{c_k} is the standard deviation at temperature c_k.

The decrement of the temperature parameter c is then determined by the user-defined parameter δ. A large value of δ induces large decrements of c, and small value of δ produces small decrements.

4.3.3 Length of Markov chains

In the SA cooling process, the length of the Markov chains must allow a significant percentage of the neighborhood S_i of a given solution $i \in S$ to be visited. The following theorem is used to quantify this percentage:

Theorem 5 *Let S be a set of cardinality $|S|$. Then, the average number of elements of S visited during a random walk with N iterations is given by:*

$$|S|.\left[1 - e^{-\frac{N}{|S|}}\right]$$

for large N and large $|S|$.

Thus, if no transition is accepted and if $N = |S_i|$, the percentage of the search space visited in the neighborhood S_i of a solution i is : $1 - e^{-1} \simeq 2/3$.

A good choice for the number of iterations of the inner loop (at temperature c_k) at iteration k is given by $L_k = |S_i|$ where, obviously, $|S_i|$ is problem dependent and to be designed by the user.

4.3.4 Stopping criterion

Let $\Delta\langle f\rangle_{c_k} = \langle f\rangle_{c_k} - f_{opt}$. Then, the execution of the algorithm should terminate when $\Delta\langle f\rangle_{c_k}$ is "sufficiently" small with respect to $\langle f\rangle_{c_0}$. For sufficiently high values of c_0, we have $< f_{c_0} > \simeq \langle f\rangle_\infty$

Moreover, for $c_k << 1$:

$$\Delta\langle f\rangle_{c_k} \simeq c_k \frac{\partial\langle f\rangle_{c_k}}{\partial c_k}$$

The end of the algorithm is then fixed by the following condition:

$$\frac{c_k}{\langle f\rangle_\infty} \frac{\partial\langle f\rangle_{c_k}}{\partial c_k} < \varepsilon_s \text{ for } c_k << 1$$

with some small tolerance ε_s to be set by the user.

4.3.5 Summary

The cooling process in polynomial time is thus parameterized by:

- The initial rate of acceptance: $\chi(c_0)$
- The distance between successive stationary distributions controlled by the parameter σ
- The stopping criterion, controlled by the parameter δ

The number of iterations of this cooling process is bounded and can be characterized by the following theorem:

Theorem 6 *Let the decrement function given by:*

$$c_{k+1} = \frac{c_k}{1 + \alpha_k c_k}$$

where

$$\alpha_k = \frac{ln(1+\delta)}{3\sigma_{c_k}}$$

And let K be the first integer for which the stopping criterion is satisfied. Then, we have $K = O(ln(|S|))$.

Consequently, if $ln(|S|)$ is polynomial on the size of the problem (which is the case for many combinatorial optimization problems), then this type of cooling induces a polynomial execution of the algorithm.

There is an optimal annealing scheme for each problem and it is up to the user to define which one is the most suitable for his application. When one has no prior information about the optimal annealing scheme, which is generally the case, one should rely on a standard geometrical scheme for which the parameter c_k evolves as follows: $c_{k+1} := \alpha_k c_k$, and tune empirically the parameters α_k and L_k on some representative instances of the class of problem of interest.

This geometric approach is not optimal for all problems but has the advantage of being robust and ensures convergence towards an approximate solution, even though it requires more time to converge than it would do with an optimal annealing scheme.

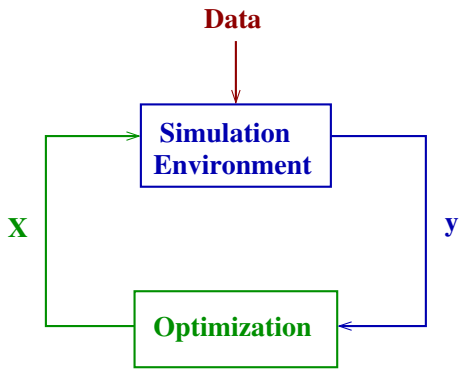

Figure 3: Objective-function evaluation based on a simulation process

4.4 Evaluation-based simulation

In many optimization applications, the objective function is evaluated thanks to a computer simulation process which requires a simulation environment. In such a case, the optimization algorithm controls the vector of decision variables, X, which are used by the simulation process in order to compute the performance (quality), y, of such decisions, as shown on Figure 3.

In this situation, population-based algorithms may not be adapted to address such problems, mainly when the simulation environment requires huge amount of memory space as is often the case in nowadays real-life complex systems. As a matter of fact, in the case of a population-based approach, the simulation environment has to be duplicated for each individual of the population of solutions, which may require an excessive amount of memory. In order to avoid this drawback, one may think about having only one simulation environment which could be used each time a point in the population has to be evaluated as follows. In order to evaluate one population, one first consider the first individual. Then, the simulation environment is initiated and the simulation associated to the first individual is run. The associated performance is then transferred to the optimization algorithm. After that, the second individual is evaluated, but the simulation environment must be first cleared from the events of the first simulation. The simulation is then run for the second individual, and so on until the last individual of the population is evaluated. In this case the memory space is not an issue anymore, but the evaluation time may be excessive and the overall process too slow, due to the fact that the simulation environment is reset at each evaluation.

In the standard simulated annealing algorithm, a copy of a state space point is requested for each proposed transition. In fact, a point $\vec{X}_j$ is gen-

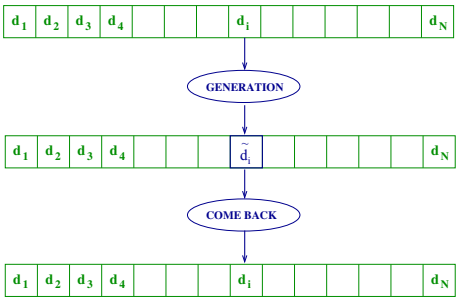

Figure 4: Optimization of the generation process. In this figure, the state space is built with a vector of decision for which the generation process consist in changing only one decision (d_i) in the current solution. If this generation, is not accepted, this component of the solution recovers its former value. The only information to be stored is the integer i and the real number d_i.

erated from the current point $\vec{X}_i$ through a copy in the memory of the computer. In the case of state spaces of large dimension, the simple process of implementing such a copy may be inefficient and may reduce drastically the performance of simulated annealing. In such a case, it is much more efficient to consider a *come back* operator, which cancels the effect of a generation. Let G be the generation operator which transform a point from $\vec{X}_i$ to $\vec{X}_j$:

$$\vec{X}_i \overset{G}{\rightarrow} \vec{X}_j$$

the comeback operator is the inverse, G_i^{-1}, of the generation operator.

Usually, such a generation modifies only one component of the current solution. In this case, the vector $\vec{X}_i$ can be modified without being duplicated. According to the value obtained when evaluating this new point, two options may be considered:

1. the new solution is accepted and, in this case, only the current objective-function value is updated.
2. else, the come back operator G^{-1} is applied to the current position in the state space in order to come back to the previous solution before the generation, again without any duplication in the memory.

This process is summarized in Figure 4.

The *come back* operator has to be used carefully because it can easely generate undesired distorsions in the way the algorithm searches the sate space. For example, if some secondary evaluation variables are used and modified for computing the overall evaluation, such variables must also recover their initial value, and the *come back* operator must therefore ensure the coherence of the state space.

5 Illustrative applications

In this section we will see how simulated annealing can be applied to two classical NP-hard combinatorial optimization problems: the knapsack problem and the traveling salesman problem.

5.1 Knapsack problem

The knapsack problem can be defined as follows. Given a set of items, each with a weight and a value, and given a capacity, determine the number of each item to include in a collection so that the associated total weight is less than or equal to the capacity, and so that the total value is as large as possible. The knapsack problem derives its name from the problem faced by someone who is constrained by a fixed-size knapsack and must fill it with the most valuable items.

This problem often arises as a subproblem in resource allocation applications where there are financial constraints, such as:

- Cargo loading (truck, boat, cargo aircraft)
- Satellite channel assignment
- Portfolio optimization

Let us consider that we have n objects that have to be put in a bag with a weight limit noted P. Each object i has a value v_i and a weight w_i. In this section, we will consider the binary version of the knapsack problem for which we must decide whether we choose an object or not (no possibility to embark multiple copies of the same item). Before presenting the application of simulated annealing to such a problem, one must first present a mathematical model for this optimization problem.

5.1.1 Mathematical Model

As for any real optimization problem to be solved, the modeling step is critical and has to be done carefully. It models the state space by defining the decision variables, and it expresses the objective function and the constraint functions in terms of the decision variables and the given data.

In the knapsack problem, the decision variables can be summarized by a binary vector x of size n for which a zero component in position i means that we leave object i, and a one means that we put object i in the bag.

For a given vector x it is quite easy to compute the associated objective function value:

$$f(x) = \sum_{i=1}^{n} v_i x_i$$

where x_i is the i^{th} component of the vector x. This represents the value of the bag, which has to be maximized.

If there were not weight limit, there would be no optimization problem in the sense that one has to put all the object in the bag in order to maximize its associated value ($x = [1,1,1,1,...,1]^T$).

The weight limitation renders the problem combinatorial. This weight limit gives the main constraint of this problem. It is modeled by the formula:

$$\sum_{i=1}^{n} w_i . x_i . \leq P.$$

Then, one must add the binary constraints:

$$x_i \in \{0,1\}, \text{ for } i = 1,2,\ldots,n$$

The overall model is then

$$\begin{gathered} \max f(x) = \sum_{i=1}^{n} v_i x_i \\ s.t. \\ \sum_{i=1}^{n} p_i x_i \leq P \\ x_i \in \{0,1\}, i = 1,2,\ldots,n \end{gathered}$$

This problem is easy to formulate but hard to solve due to the associated combinatorics. For n objects, the number of potential solutions to consider is 2^n which grows very rapidly with n:

n	2^n
10	$1.024\ 10^3$
20	$1.048\ 10^6$
30	$1.073\ 10^9$
40	$1.099\ 10^{12}$
50	$1.125\ 10^{15}$
60	$1.152\ 10^{18}$
70	$1.180\ 10^{21}$
80	$1.208\ 10^{24}$
90	$1.237\ 10^{27}$
100	$1.267\ 10^{30}$

For large instances of the knapsack problem, one can consider applying metaheuristics like simulated annealing.

5.1.2 Simulated Annealing Implementation

For the knapsack problem, each solution is encoded as a binary vector, X.

From a point X_i we generate a neighbor X_j by randomly flipping one component of X_i, as shown on Figure 5 where the k^{th} component is chosen.

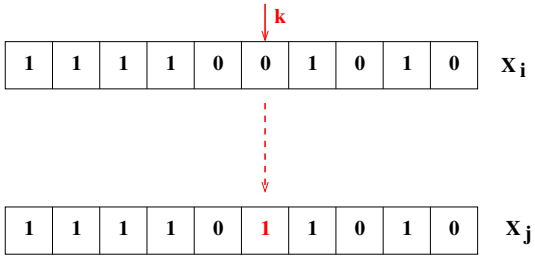

Figure 5: In this example, with $n = 10$, the sixth position has been randomly selected in order to include the sixth object in the bag.

In the unconstrained optimization context of SA, a classical relaxation can be considered to take into account the capacity constraint. Basically, a term is added in the objective function to penalize the violation of this constraint. Here, we compute the weight excess Δ when the weight of the items in the knapsack exceeds its capacity:

$$\Delta = \min(0, (\sum_{i=1}^{n} w_i x_i) - P)$$

and the objective function value is then penalized by subtracting from it $\mu \frac{\Delta}{P}$, where μ is a penalty parameter to be set by the user.

In order to test the simulated annealing algorithm on this problem, we first build an instance of the problem by randomly generating 100 objects for which the weights have also been selected randomly between 1 and 100 with a uniform probability density function. For this instance, the capacity of the bag is set to $P = 2000$. We choose $\mu = 1$ for the penalty parameter and we apply the basic SA algorithm with the initial temperature set to a value of c_0 such that $\chi(c) = 0.8$, a geometric cooling schedule with $\alpha = 0.995$, and $L_k = 1,000$ for every iteration k. The algorithm is stopped when the temperature reaches $\frac{c_0}{1,000}$.

We propose as initial solution a uniformly-distributed random binary vector. The evolution of the penalized objective function with the number of iterations is shown in Figure 6, and the associated evolution of the total weight and the value of the knapsack is shown in Figure 7. At the beginning of the optimization process, the SA explores the solution space by accepting solutions that yield low value of the penalized objective function. This leads to high excess weight and high total value. The value of the penalized objective function increases as the algorithm converges to the optimal solution. Since the excess weight is high at the beginning, the solution is improved mainly by removing weight from the knapsack, therefore the total weight and total value decrease. As the excess weight reaches zero, the solution must be improved by decreasing weight (therefore more item can be taken) and also increase the value of each taken item. Therefore, the total value increases until it reaches the maximum value.

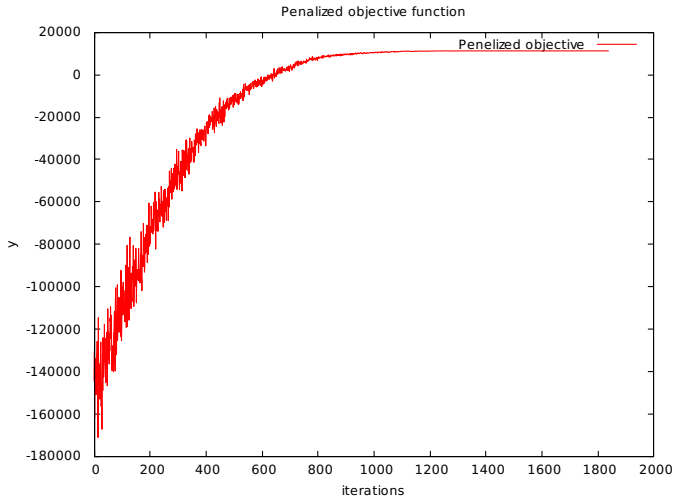

Figure 6: Evolution of the penalized objective function with iterations

5.2 Traveling Salesman Problem

The traveling salesman problem (TSP) asks the following question: "Given a list of n cities, among which an origin city, and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city?" This is again an important NP-hard combinatorial optimization problem, particularly in the fields of

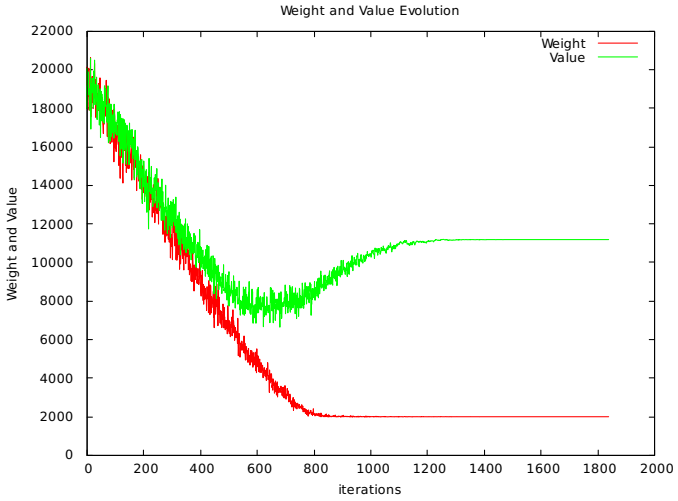

Figure 7: Evolution of the total weight and value with iterations

operations research and theoretical computer science. The problem was first formulated in 1930 and is one of the most intensively-studied problems in discrete optimization.

Let us consider a set of n cities where each city i has coordinates $(x_i, y_i), i = 1, 2, \ldots, n$.

As for the knapsack problem, a mathematical model is first presented.

5.2.1 Mathematical Model

In this case, each point X of the state space, has to represent a potential permutation in the order we visit the n cities. For simplicity, we consider the following initial solution using the lexicographic order:

$$X_0 = \boxed{1}\boxed{2}\boxed{3}\boxed{4}\boxed{\ldots}\boxed{n}$$

The objective function evaluation consists in computing the length f of the tour corresponding to any vector X:

$$f(X) = \sum_{i=1}^{n-1} d(X_i, X_{i+1}) + d(X_N, X_1)$$

where, X_i is the i^{th} element of X. If $X_i = k$ and $X_{i+1} = l$, the inter-city distance is:

$$d(X_i, X_{i+1}) = \sqrt{(x_l - x_k)^2 + (y_l - y_k)^2}$$

Note that the last term, $d(X_N, X_1)$, in the above definition of f represents the last segment of the tour to come back to the origin city.

The complexity associated with the traveling salesman problem is known to be much higher than that of the knapsack problem. For a problem with n cities, the number of potential tours to be considered is $n!$, which grows with n much faster than 2^n:

n	2^n	n!
10	$1.024\ 10^3$	$3.628\ 10^6$
20	$1.048\ 10^6$	$2.432\ 10^{18}$
30	$1.073\ 10^9$	$2.652\ 10^{32}$
40	$1.099\ 10^{12}$	$8.159\ 10^{47}$
50	$1.125\ 10^{15}$	$3.041\ 10^{64}$
60	$1.152\ 10^{18}$	$8.320\ 10^{81}$
70	$1.180\ 10^{21}$	$1.197\ 10^{100}$
80	$1.208\ 10^{24}$	$7.156\ 10^{118}$
90	$1.237\ 10^{27}$	$1.485\ 10^{138}$
100	$1.267\ 10^{30}$	$9.332\ 10^{157}$

Just to give an idea of the complexity of the problem, if one evaluation of the objective function requests 10^{-9} seconds, then a naive enumeration algorithm evaluating every possible solution would require the following CPU time:

n	2^n	$n!$	ratio $\frac{n!}{2^n}$
10	1 micro second	3.6 mili seconds	$3.6\ 10^3$
20	1 mili second	77 years	$2.3\ 10^{12}$
30	1 second	$8.4\ 10^{15}$ years	$2.47\ 10^{23}$
40	18 minutes	$2.5\ 10^{31}$ years	$7.4\ 10^{35}$
50	13 days	$9.6\ 10^{47}$ years	$2.7\ 10^{49}$
60	36 years	$2.6\ 10^{47}$ years	$7.2\ 10^{63}$
70	$37\ 10^3$ years	$3.8\ 10^{83}$ years	$1\ 10^{79}$
80	$38\ 10^6$ years	$2.2\ 10^{102}$ years	$5.9\ 10^{94}$
90	$39\ 10^9$ years	$4.7\ 10^{121}$ years	$1.2\ 10^{111}$
100	$40\ 10^{12}$ years	$2.9\ 10^{141}$ years	$7.3\ 10^{127}$

Even if the computer power is likely to double in the next 18 months, no need to say that it would not make such a naive algorithm practical.

5.2.2 Simulated Annealing Implementation

One of the simplest neighborhood operator for this problem consists of randomly exchanging two positions in the current solution vector X (see Fig-

ure 8). This way of manipulating points of the state space ensures that the produced neighbor remains a permutation *i.e.* a tour of the n cities. Implementing such an operator within the SA algorithm yields acceptable

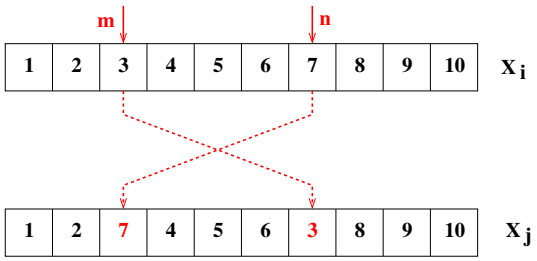

Figure 8: A first neighborhood operator: randomly swapping two positions.

results, but the performance of the SA can really be improved by using a neighborhood operator that exchanges all the positions between two randomly chosen indices (m,n), as shown in Figure 9.

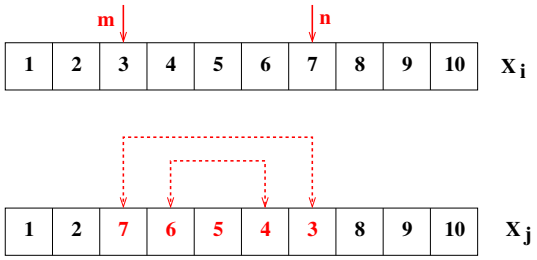

Figure 9: A second neighborhood operator: swapping all positions between two randomly chosen positions (m,n).

Let us consider an instance with $n = 1000$ cities randomly generated in a square subset of the plane. The straightforward SA algorithm is implemented, again, with initial temperature c_0 such that $\chi(X) = 0.8$, a geometric cooling schedule with $\alpha = 0.995$, and $L_k = 1,000$ for every iteration. The algorithm is stopped when the temperature reaches $\frac{c_o}{1,000}$, and based on the second neighborhood operator (Figure 9). The initial solution considered is the tour of total distance $1.16857164.10^8$ shown in Figure 10.

After application of the simulated annealing algorithm on this problem, one obtains the tour displayed in Figure 11. One minute of computation on a Unix platform with a 2.4 GHz processor and 8 GB of RAM was needed to get the final tour of total distance 360,482.

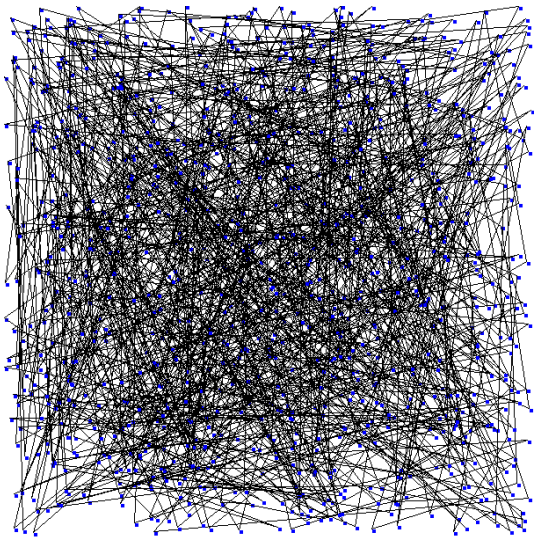

Figure 10: Initial tour of the TSP with $n = 1,000$ cities.

This is clearly not an optimal solution for this instance (there are some suboptimal crossings) but this solution is very easily obtained via a direct application of SA.

Simulated annealing has also been applied to many combinatorial problems coming from the industry and real-world operations. To mention just a few:

- Airline Crew Scheduling [8]
- Railway Crew Scheduling [9]
- Traveling Salesman Problem [4]
- Vehicle Routing Problem [14]
- Layout-Routing of Electronic Circuits [17]
- Large Scale Aircraft Trajectory Planing [5, 10]
- Complex portfolio problem [7]
- Graph coloring problem [6]
- High-dimensionality minimization problems [16]

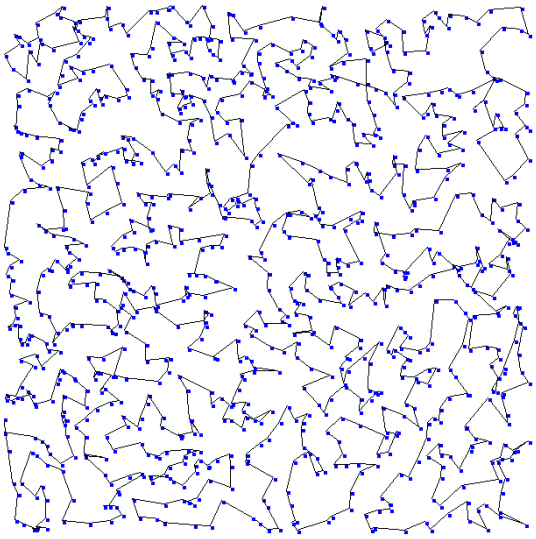

Figure 11: Final tour of the TSP with $n = 1,000$ cities.

6 Large-scale aircraft trajectory planning

In this section, we present a methodology using SA to address a strategic planning of aircraft trajectories at the European continental scale, which involves nearly 30,000 flights per day. The goal is to separate the given set of 4D aircraft trajectories (three-dimension space plus time) by allocating an alternative route in the three-dimension space and an alternative departure time to each flight.

6.1 Mathematical model

Our strategic trajectory planning problem considers a set of flight plans (origin, destination, departure time) for a given day. We rely on route or departure-time allocation to separate aircraft trajectories. In other words, for each flight, we can delay departure and/or impose an alternative route instead of the initially-planned direct route between the origin and the destination. This can be formulated as an optimization problem aimed at minimizing the number of *interactions* between trajectories, where we count one interaction whenever two flights are *in conflict* i.e., separated at some point by less than 5 NM (nautical miles) horizontally or 1,000 feet vertically.

Given data. A problem instance is given by:

- A set of N initial (nominal) discretized 4D (direct-route) trajectories;
- For each flight i, for $i = 1, 2, \ldots, N$:
  - The initial planned departure time: $t_{i,0}$;
  - The maximum allowed advance departure time shift: $\delta_a^i < 0$;
  - The maximum allowed delay departure time shift: $\delta_d^i > 0$;
  - The maximum allowed route length extension coefficient: $0 \leq d_i \leq 1$.
  - M: the number of allowed virtual waypoints to modify the route.

Decision Variables. In the time domain, one can use a departure-time shift, δ_i, associated with each flight i ($i = 1, 2, \ldots, N$). Therefore, the resulting departure time of flight i is given by $t_i = t_{i,0} + \delta_i$, In the 3D space, one can rely on a vector, w_i, of virtual *waypoint locations* through which flight i must go (using straight-line segments), $w_i := (w_i^1, w_i^2, \ldots, w_i^M), i = 1, \ldots, N$. Let us set the compact vector notation: $\delta := (\delta_1, \delta_2, \ldots, \delta_N)$, and $\mathbf{w} := (w_1, w_2, \ldots, w_N)$. Therefore, the decision variables of our route / departure-time allocation problem can be represented by the vector: $u := (\delta, \mathbf{w})$.

Constraints. The above optimization variables must satisfy the following constraints:

- ***Allowed departure time shift***. Since it is not reasonable to delay or to advance departure times for too long, the departure time shift, δ_i, is limited to lie in the interval $[\delta_a^i, \delta_d^i]$. Common practice in airports led

us to discretize this time interval. Given the (user-defined) time-shift step size δ_s, this yields $N_a^i := \frac{|\delta_a^i|}{\delta_s}$ possible advance slots, and $N_d^i := \frac{\delta_d^i}{\delta_s}$ possible delay slots for flight i. Therefore, we define the discrete set, Δ_i, of all possible departure time shifts for flight i by

$$\Delta_i := \{-N_a^i.\delta_s, -(N_a^i-1).\delta_s, \ldots, -\delta_s, 0, \delta_s, \ldots, (N_d^i-1).\delta_s, N_d^i.\delta_s\}. \tag{7}$$

- ***Maximal route length extension***. The alternative trajectory to be chosen increases the route length, which leads to an increase in fuel consumption and flight time. Therefore, the alternative choice should be limited for the new trajectory if it is to be accepted by the airline. Consequently, the alternative trajectory for flight i must satisfy:

$$L_i(w_i) \leq (1+d_i), \tag{8}$$

  where $L_i(w_i)$ denotes the *normalized length* (*i.e.*, assuming that the direct-flight path length is 1) of the alternative trajectory determined by the waypoint vector w_i.

- ***Allowed waypoint locations.*** To reduce the search space, prevent undesirable sharp turns, and restrain the route length extension, we bound the possible location of each virtual waypoint. Let W_{ix}^m and W_{iy}^m be the 2D sets of all possible normalized longitudinal and lateral locations, respectively, of the m^{th} virtual waypoint for trajectory i. The (normalized) longitudinal component, w_{ix}^m, must lie in the interval:

$$W_{ix}^m := \left[\left(\frac{m}{1+M} - b_i\right), \left(\frac{m}{1+M} + b_i\right)\right], m = 1, 2, \ldots, M \tag{9}$$

  where $0 \leq b_i \leq 1$ is a (user-defined) model parameter. The normalized lateral component, w_{iy}^m, is restricted to lie in the interval:

$$W_{iy}^m := [-a_i, a_i], \tag{10}$$

  where $0 \leq a_i \leq 1$ is a (user-defined) model parameter chosen *a priori* so as to satisfy (8). This yields a rectangular shape for the possible locations of the virtual waypoint w_i^m (see Figure 12).

Objective function The objective is to minimize the number of *interactions between trajectories*, which correspond, roughly speaking, to situations that occur in the flight planning phase, when more than one trajectory compete for the same space at the same period of time. Consider for example the trajectories A, B and C in Figure 13. We define an *interaction at a trajectory point* $P_{i,k}(u_i)$ to be the sum of all the conflicts associated

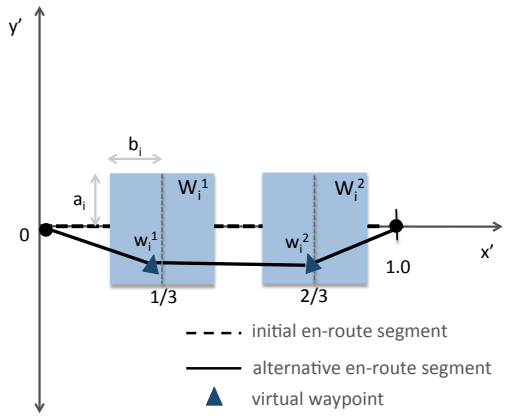

Figure 12: Rectangular-shape sets of the possible locations of $M = 2$ virtual waypoints, for trajectory i.

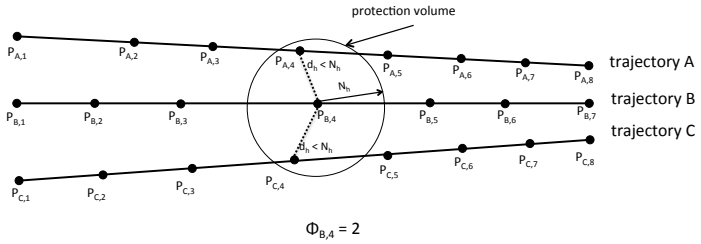

Figure 13: Interactions, $\Phi_{B,4}$, at sampling point $P_{B,4}$ of trajectory B.

with point $P_{i,k}(u_i)$, where u_i the i^{th} component of u. We further define the *interaction,* Φ_i, *associated with trajectory* i, as: $\Phi_i(u) := \sum_{k=1}^{K_i} \Phi_{i,k}(u)$ where K_i is the number of trajectory points obtained through some discretization of the trajectory of the i^{th} flight. Figure 13 illustrates the case of trajectory $i = B$ at the trajectory point $P_{B,4}$. Finally, *interaction between trajectories*, Φ_{tot}, for a whole traffic situation is simply defined as:

$$\Phi_{tot}(u) := \sum_{i=1}^{N} \Phi_i(u) = \sum_{i=1}^{N} \sum_{k=1}^{K_i} \Phi_{i,k}(u). \tag{11}$$

The interaction minimization problem can be formulated as a mixed-integer optimization problem, as follows:

$$\min_{u=(\delta,\mathbf{w})} \Phi_{tot}(u)$$

subject to

$$\begin{aligned} &\delta_i \in \Delta_i, \quad \text{for all } i = 1,2,\ldots,N \\ &w_{ix}^m \in W_{ix}^m, \quad \text{for all } i = 1,2,\ldots,N, m = 1,2,\ldots,M \\ &w_{iy}^m \in W_{iy}^m, \quad \text{for all } i = 1,2,\ldots,N, m = 1,2,\ldots,M, \end{aligned} \tag{P1}$$

where the set Δ_i is defined in (7), and W_{ix}^m and W_{iy}^m are defined in (9) and (10), respectively.

In order to evaluate the objective function of a candidate solution, $(\mathbf{w}, \delta)$, one needs to compute the interaction, Φ_{tot}, between the N aircraft trajectories. To avoid the $\frac{N(N-1)}{2}$ time-consuming pair-wise comparisons, which is prohibitive in our large-scale application context, we propose a 4D grid-based conflict detection scheme as illustrated in Figure 14 (see [5, 10] for further details). First, we define a four-dimensional (3D space + time) grid (see Figure 14). The size of each cell in the x, y, and z directions is defined by the minimum separation requirements, $N_h = 5$ NM and $N_v = 1{,}000$ feet. The size of the cell in the time domain is set according to some given discretization step size, t_s. To detect conflicts, the idea is to successively put each trajectory in this grid, and then check for conflicts only in the cells surrounding the current trajectory.

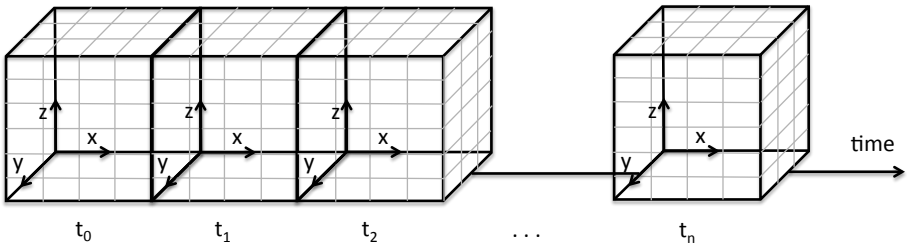

Figure 14: Four dimension (space - time) grid.

In the SA optimization process, the computation of the objective function, $\Phi_{tot}(u)$, is repeated many times. Therefore it must be computed as efficiently as possible. To avoid checking interactions over all the N trajectories even when only a subset of trajectories are modified in a new proposed solution, the interaction count is updated in a *differential* manner. More precisely, we proceed as follows. First, the 4D grid is initialized with every cell empty. Then, the initial N trajectories, corresponding to the initial value of the decision vector, u (with all its components at zero, i.e., direct flight), are placed in the 4D grid and the *current* interaction, Φ_{iC}, associated with each trajectory, i, and the current total interaction between trajectories, Φ_{totC}, are computed.

We assume now that during the optimization process, the decision variables of l flights are to be modified. Let I_{modif} be a list of length l containing the flight indices of the l flights. To update the value of total interaction, we first remove all the l corresponding trajectories from the 4D grid. Therefore, the interaction associated with each trajectory in I_{modif} is set to an intermediate value $\Phi_{i,inter}(u) = 0, \quad \forall i \in I_{modif}$. It should be noted that the interaction measurement is symmetrical: if $\Phi^{ij}(u)$ denotes the *contribution of trajectory i to the interaction associated with trajectory j*, then $\Phi^{ij}(u) = \Phi^{ji}(u)$. Let $\mathcal{N}_i$ be a set of trajectories currently interacting with trajectory i. The interaction associated with trajectory $j \in \mathcal{N}_i$ over all trajectories $i \in I_{modif}$, is set to an intermediate value $\Phi_{j,inter}(u) = \Phi_j(u) - \sum_{i \in I_{modif}} \Phi^{ij}(u)$. Thereafter, the *modified* trajectories corresponding to the new decision variable values, u_i, $i \in I_{modif}$, are placed in the 4D grid and the interaction detection procedure is performed over all trajectories $i \in I_{modif}$. Then, the interaction, Φ_i, associated with each trajectory $i \in I_{modif}$, is computed. Again, the interaction associated with each trajectory, j, interacting with the set of modified trajectories is updated as follows: $\Phi_j(u) = \Phi_{j,inter}(u) + \sum_{i \in I_{modif}} \Phi^{ij}(u)$. Finally, the total interaction between trajectories is simply computed as $\Phi_{tot}(u) = \sum_{i=1}^{N} \Phi_i(u)$. This interaction computation method allows us to update the value of the objective function when some trajectories are modified within a very short computation time, since we do not need to compute the change of interaction for decisions that are not modified at the current optimization iteration.

6.2 Computational experiments with SA

The proposed methodology is tested with a continent-size air traffic instance for a full day of air-traffic over the European airspace, consisting of $N = 29{,}852$ en-route trajectories. The trajectories are sampled with a discretization step of t_s = 20 seconds. The initial trajectory set involves $\Phi_{tot} = 142{,}144$ total interactions between trajectories. Figure 15 illustrates the initial trajectory points (blue dots), and the locations where the initial interactions occur (red dots).

The initial temperature is computed by first generating 100 deteriorating transformations at random and then by evaluating the average variations, $\Delta\Phi_{avg}$, of the objective function values. The initial temperature, c_0, is then deduced from the relation: $c_0 = e^{\frac{\Delta\Phi_{avg}}{T_0}}$, where τ_0 is the initial acceptance rate of degrading solutions (which will be empirically set). In order to reach an equilibrium, a sufficient number of iterations, denoted L_k, have to be performed at each temperature step k. In our case, we assume for simplicity purposes that the number of iterations, L_k, is constant and empirically set. The temperature is decreased following the geometrical law, $c_{k+1} = \alpha c_k$, where $0 \leq \alpha \leq 1$ is a pre-defined constant value.

To generate a solution in the neighborhood, we set a user-defined threshold value of interaction, denoted Φ_τ, such that the trajectory of a randomly chosen flight i will

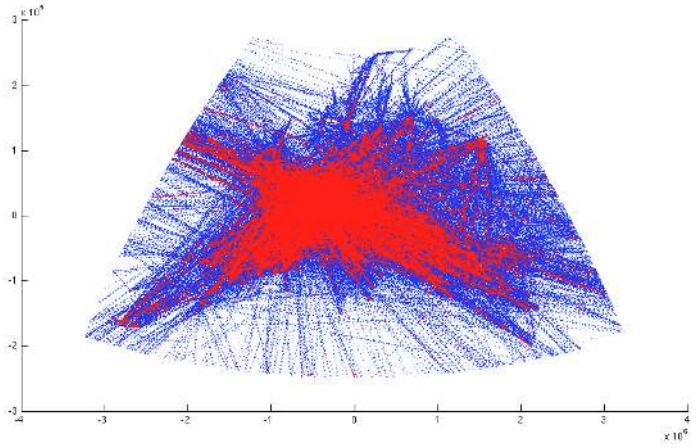

Figure 15: Initial (direct-route) trajectory set involving one-day en-route air traffic over the European airspace (29,852 flights) sampled with t_s = 20 seconds with initial location of interactions displayed as red color dots.

be modified only if $\Phi_i(u) \geq \Phi_\tau$, where u is the current solution. Then, for a chosen flight, i, we introduce another user-defined parameter, $P_w \leq 1$, to control the probability of modifying the value of the i^{th} trajectory waypoint location decision vector, w_i. The probability to modify instead the departure time is thus $1 - P_w$. The algorithm terminates when the final temperature, c_f, is reached, or when an interaction-free solution is found. The parameter values chosen to specify the instance considered, and the empirically set parameters defining the overall SA problem-solving methodology are given in Table 1.

The SA adapted to solve the strategic trajectory planning problem is implemented in Java. We address this problem instance with an AMD Opteron 2 GHz processor with 128 Gb RAM. Numerical results obtained from the simulation are reported in Table 2. This SA implementation yields an interaction-free solution for this continent-scale problem instance after around 76 minutes of computation time. This is compatible with strategic (several days in advance) planning application requirements in the setting of regular airline schedules.

Parameters defining the Problem		Parameters defining the SA	
parameter	value	parameter	value
$-\delta_a^i = \delta_d^i$	60 minutes	L_k	3,500
δ_s	20 seconds	τ_0	0.3
d_i	0.12 (12 %)	β	0.99
M	2	T_f	$(1/500).T_0$
a_i	0.126	P_w	0.5
b_i	0.067	Φ_τ	0.5 Φ_{avg}

Table 1: Chosen (user-defined) parameter values defining the problem and the empirically-set (user-defined) parameter values of the resolution methodology

Numerical results	value
number of iterations	497,000
avg. computation time (minutes)	76.19
avg. proportion of delayed / advanced flights	71.29%
avg. proportion of extended flights	46.23%
avg. departure time shifts (minutes)	30.14
avg. route length extensions	1.95%

Table 2: Numerical results for continent-size problem instance solved by SA (averages are computed over 10 runs).

7 Conclusion

This chapter introduced the reader to simulated annealing (SA), a global optimization metaheuristic. The main advantage of SA is its simplicity. SA is based on an analogy with the physical annealing of materials that avoids the drawback of the Monte-Carlo approach (which can be trapped in local minima), thanks to an efficient Metropolis acceptance criterion. When the objective function evaluations require a lot of memory space, for example when it results from complex simulation processes that manipulate large-dimension state space involving much memory, population-based algorithms are not applicable and simulated annealing is the right answer to address such issues. An illustration was provided in section 6 where a large-scale complex aircraft trajectory planning problem involving nearly 30,000 flights over Europe was addressed by exploiting particular features of the problem and, in particular, by integrating clever implementation techniques within the algorithm, and by setting user-defined parameters empirically, along the lines of the basic SA theory.

References

[1] E. Aarts and J. Korst. *Simulated Annealing and Boltzmann Machines: A Stochastic Approach to Combinatorial Optimization and Neural Computing*. Wiley, NY, USA, 1989.

[2] E. Aarts and P. Van Laarhoven. A new polynomial time cooling schedule. In *Proceedings of the IEEE International Conference on Computer-Aided Design, Santa Clara*, pages 206–208.

[3] E. Aarts and P. Van Laarhoven. Statistical cooling : A general approach to combinatorial problems. *Philips Journal of Research*, 40:193–226, 1985.

[4] H. Bayram and R. Sahin. A new simulated annealing approach for travelling salesman problem. *Mathematical and Computational Applications*, 18(3):313–322, 2013.

[5] S. Chaimatanan, D. Delahaye, and M. Mongeau. A hybrid metaheuristic optimization algorithm for strategic planning of 4D aircraft trajectories at the continental scale. *IEEE Computational Intelligence Magazine*, 9(4):46–61, 2014.

[6] M. Chams, A. Hertz, and D. de Werra. Some experiments with simulated annealing for coloring graphs. *European Journal of Operational Research*, 32(2):260 – 266, 1987.

[7] Y. Crama and M. Schyns. Simulated annealing for complex portfolio selection problems. *European Journal of Operational Research*, 150(3):546–571, 2003.

[8] T. Emden-Weiner and M. Proksch. Best practice simulated annealing for the airline crew scheduling problem. *Journal of Heuristics*, 5(4):419–436, 1999.

[9] R. Hanafi and E Kozan. A hybrid constructive heuristic and simulated annealing for railway crew scheduling. *Computers & Industrial Engineering*, 70:11–19, 2014.

[10] A. Islami, S. Chaimatanan, and D. Delahaye. Large-scale 4D trajectory planning. In Electronic Navigation Research Institute, editor, *Air Traffic Management and Systems II*, volume 420 of *Lecture Notes in Electrical Engineering*, pages 27–47. Springer Japan, 2017.

[11] S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671, 1983.

[12] S. Kirkpatrick, D. Gelatt, C, and M.P. Vecchi. Optimization by simulated annealing. IBM Research Report RC 9355, Acts of PTRC Summer Annual Meeting, 1982.

[13] P. Laarhoven and E. Aarts, editors. *Simulated Annealing: Theory and Applications*. Kluwer, Norwell, MA, USA, 1987.

[14] W.F. Mahmudy. Improved simulated annealing for optimization of vehicle routing problem with time windows (VRPTW). *Kursor Journal*, 7(3):109–116, 2014.

[15] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller. Equation of state calculation by fast computing machines. *Journal of Chemical Physics*, 21(6):1087–1092, 1953.

[16] P. Siarry, G. Berthiau, F. Durdin, and J. Haussy. Enhanced simulated annealing for globally minimizing functions of many continuous variables. *ACM Transactions on Mathematical Software*, 23(2):209–228, 1997.

[17] D.F. Wong, H.W. Leong, and C.L. Liu. *Simulated annealing for VLSI design*. Kluwer, 1988.