Stochastic Methods for Hard Optimization. Application to Robust Fault Diagnosis and Control of Industrial Systems

Rosario Toscano
Université de Lyon, Laboratoire de Tribologie et de Dynamique des Systèmes
Ecole Nationale d'Ingénieurs de Saint-Etienne, France

Abstract. This chapter aims at solving difficult optimization problems arising in many engineering areas. To this end, a brief review of the main stochastic methods which can be used for solving continuous non-convex constrained optimization problems is presented i.e.: Simulated annealing (SA), Genetic algorithm (GA), and Particle swarm optimization (PSO). In addition to that, we will present a recently developed optimization method called Heuristic Kalman Algorithm (HKA) which seems to be, in some cases, an interesting alternative to the conventional approaches. The stochastic methods SA, GA, PSO and HKA, will be compared through various numerical experiments. The performance of these methods depends dramatically on the feasible search domain used to find out a solution as well as the initialization of the various user defined parameters. From this point of view, some practical indications concerning these issues will be given. Another objective of this chapter is to show that the stochastic methods, notably HKA, can be efficiently used to solve robust synthesis problems in the area of structured control and fault diagnosis systems. More precisely, we will deal with the following problems: the synthesis of a robust controller with a given fixed structure and the design of a robust residual generator. Some numerical experiments exemplify the resolution of this kind of problems.

Keywords. Non-convex optimization problems, stochastic optimization methods, metaheuristic, simulated annealing, genetic algorithm, swarm intelligence, particle swarm optimization, heuristic Kalman algorithm, parameter setting, feasible domain, robust control, fixed structure controller, PID controller, robust $\mathscr{H}_\infty$ control, optimal residual generator, mixed $\mathscr{H}_2/\mathscr{H}_\infty$ optimization problem, robust fault detection.

INTRODUCTION

In all areas of engineering, physical and social sciences, one encounters problems involving the optimization of some objective function. Usually, the problem to solve can be formulated precisely but is often, difficult or impossible to solve either analytically or through conventional numerical procedures. This is the case when the problem is non-convex and so inherently nonlinear and multimodal. In fact it is now well established that the frontier between the efficiently solvable optimization problems and the others rely on its convexity (Rockafellar, 1993). This is confirmed by the fact that very efficient algorithms for solving convex problems exist (Boyd & Vandenberghe, 2004), whereas the problem of non-convex optimization remains largely open despite an enormous amount of effort devoted to its resolution.

In this context, several stochastic methods, also called metaheuristics, have been developed in the last two decades, which have demonstrated a strong ability to solve problems that were previously difficult or impossible to solve (Fogel 2006; Goldberg, 1989; Kennedy & Eberhart, 1995; Kirkpatrick, Gelatt & Vecchi, 1983; Spall, 2003). These metaheuristics include simulated annealing (SA), genetic algorithm (GA), and particle swarm (PS), to cite only the most used in the framework of continuous optimization problems. The main characteristic of these approaches is the use of a stochastic mechanism for seeking a solution. From a general point of view, the use of a stochastic search procedure seems in fact unavoidable in finding a promising solution of non-convex optimization problems. Since this kind of difficult optimization problem is frequently encountered in practice, it is very important to give an exploitable material for engineers who have to design optimal systems. This is also true for academic researchers who are often faced with the challenge of solving non-convex optimization problems.

With this in mind, one of the objectives of this chapter is to give a brief review of the main stochastic methods which can be used for solving continuous non-convex constrained optimization problems i.e.: Simulated annealing (SA), Genetic algorithm (GA), and Particle swarm optimization (PSO). Although a large number of approaches have been proposed in the literature to improve these stochastic methods, non-convex optimization is still a challenging subject, mainly because of very large variability concerning the topological properties of the underlying objective function. For this reason, it is always useful to explore new principles allowing the resolution of a wide range of non-convex optimization problems. In this spirit, another objective of this chapter is to introduce a new alternative optimization method (developed by the author), which we call Heuristic Kalman Algorithm (HKA) (Toscano & Lyonnet, 2009a). The stochastic methods SA, GA, PSO and HKA, will be compared through various numerical experiments. The performance of these methods depends dramatically on the feasible search domain used to find out a solution as well as the initialization of the various user defined parameters. From this point of view, some practical indications concerning these issues will be given. In particular, each optimization method will be accompanied with its specific parameter setting.

Another objective of this chapter is to show that the stochastic methods, notably HKA, can be efficiently used to solve robust synthesis problems in the area of structured control and fault diagnosis systems. More precisely, we will deal with the following problems: the synthesis of a robust controller with a given fixed structure (e.g. MIMO PID) and the design of a robust residual generator. The main motivation for considering this kind of problems is that they require the resolution of non-convex optimization problems, which are difficult to solve via usual methods.

BACKGROUND: THE OPTIMIZATION PROBLEM

Optimization is the way of obtaining the best possible outcome given the degrees of freedom and the constraints. To make our discussion more precise, consider the general system presented in Figure 1, which produces an output in response to a given input. In addition, this system has some tuning parameters allowing the modification of its behaviour. By behavior we mean the relationship existing between the inputs and outputs.

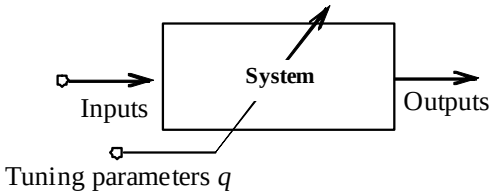

Figure 1 – *An optimization problem.*

The problem is then how to tune these parameters so that the system behaves well. Usually, the desired behavior can be formulated via an *objective function* (or *cost function*) depending on the tuning parameters $f(q)$, which needs to be maximized or minimized with respect to q. More formally, the problem to solve can be formulated as follows: find the optimal tuning parameters q_{opt}, solution of the following problem:

$$\begin{cases} q_{opt} = \arg\min_{q \in \mathcal{F}} f(q) \\ \mathcal{F} = \left\{ q \in \mathcal{D} : g_i(q) \le 0, i = 1, \ldots, N_c \right\} \\ \mathcal{D} = \left\{ q \in \mathbf{R}^{n_q} : \underline{q} \le_e q \le_e \overline{q} \right\} \end{cases} \quad (1)$$

where $f : \mathbf{R}^{n_q} \to \mathbf{R}$ is a function for which the minimum[1] ensures that the system behaves as we want, $\mathcal{F}$ is the feasible domain i.e. the set of vector $q \in \mathcal{D}$ satisfying the N_c constraints g_i, and $\mathcal{D}$ is the search domain[2] i.e. the set under which the minimization is performed. Generally $q = [q_1 \cdots q_{n_q}]^T$ is called the *design* (or *decision*) vector, and its n_q components the decision or design variables. The vectors $\underline{q} = [\underline{q}_1 \cdots \underline{q}_{n_q}]^T$ and $\overline{q} = [\overline{q}_1 \cdots \overline{q}_{n_q}]^T$ are the bounds of the search domain and the symbol $\le_e$ means a componentwise inequality. The functional constraints g_i can be handled by introducing a new objective function including such is called penalty functions:

$$J(q) = f(q) + \sum_{i=1}^{N_c} w_i \max(g_i(q), 0) \quad (2)$$

Where N_c is the number of constraints and the w_i's are weighting factors. The setting of the w_i's is not very critical, it is only required to penalize more or less strongly the violation constraints. Note that if q satisfy the constraints then $J(q) = f(q)$. In these conditions solving problem (1) is the same as solving the following optimization problem:

$$\begin{cases} q_{opt} = \arg\min_{q \in \mathcal{D}} J(q) \\ \mathcal{D} = \left\{ q \in \mathbf{R}^{n_q} : \underline{q} \le_e q \le_e \overline{q} \right\} \end{cases} \quad (3)$$

Thus posed, the objective is then to find the optimum q_{opt} i.e. the n_q-dimensional decision vector $q \in \mathcal{D}$ which minimizes the cost function J.

Unfortunately, there are several obstacles for solving this kind of problem. The main obstacle is that most of the optimization problems are NP-hard (Garey & Johnson, 1979). Therefore the known theoretical methods cannot be applied except possibly for some small size problems. Other difficulties are that the cost function may be not differentiable and/or multimodal. Therefore the set of methods requiring the derivatives of the cost function cannot be used. Another obstacle is when the cost function cannot be expressed in an analytic form, in this case, the cost function can be only evaluated through simulations.

---

[1] Note that any maximisation problem can be converted into a minimization problem, indeed: $q_{opt} = \arg\max_{q \in \mathcal{F}} J(q) = \arg\min_{q \in \mathcal{F}} - J(q)$

[2] $\mathcal{D}$ is a hyberbox and so it is also called the hyperbox search domain.

In these situations, heuristic approaches seem to be the only way for solving optimization problems. By heuristic approach, we mean a computational method employing experimentations, evaluations and trial-and-errors procedures in order to obtain an approximate solution for computationally difficult problems. In the next sections we will review some standard heuristic approaches for solving the optimization problem (3), namely: simulated annealing (SA), genetic algorithm (GA) and particle swarm optimization (PSO). These methods are indeed the most widely used in the context of continuous optimization which is the scope of this chapter. In addition to that, we will present a recently developed optimization method called Heuristic Kalman Algorithm (HKA) which seems to be, in some cases, an interesting alternative to the conventional approaches.

STOCHASTIC METHODS FOR SOLVING HARD OPTIMIZATION PROBLEMS

1. Simulated annealing

Simulated annealing (SA) is a random optimization method introduced by S. Kirkpatrick in 1983 and by V. Cerný in 1985 (Kirkpatrick, Gelatt & Vecchi, 1983; Cerny, 1985). The name comes from a technique used in metallurgy, called annealing, which consists in heating and slowly cooling a metal to obtain a "well ordered" solid state of minimal energy (Dréo *et al.*, 2006). More precisely, the annealing consists in lowering the temperature gradually, *in stage*, allowing to obtain, at each stage, a thermal equilibrium. At high temperatures, atoms are very mobile, but as the temperature decreases this mobility is diminished and the atoms tend to form a solid structure with lower internal energy than the initial one. To achieve a minimum-energy state, the cooling must occur at a sufficiently slow rate. If the temperature of the substance is decreased too rapidly, an amorphous or polycrystalline structure may be obtained which is not a minimum-energy state of the substance.

Simulated annealing is based upon the *Metropolis algorithm* (Metropolis *et al.*, 1953), which was originally proposed as a means of finding the equilibrium configuration of a collection of atoms at a given temperature. The connection between this algorithm and mathematical minimization was first noted by Pincus (1970), but it was Kirkpatrick *et al.* (1983) who proposed that it form the basis of an optimization technique for combinatorial problems. The approach has been later extended to continuous global optimization problems of type (3). Indeed, each point q of the search space $\mathcal{D}$ can be seen as a state of some physical system, and the function $J(q)$ to be minimized would represent the internal energy of the system in that state. Thus, searching for an optimal solution is like finding a configuration of the cooled system with minimum internal energy. The aim of SA is then to bring the "system", from an arbitrary *initial state*, to a state of minimal energy i.e. minimal J. An interesting property of SA is its ability to avoid getting stuck in local minima (Corana *et al.* 1987). This is obtained by using a random procedure which not only accepts changes that decrease the cost function J (assuming a minimization problem), but also some changes that increase it. The latter are accepted in accordance with a probabilistic rule known as the *Metropolis criterion* (see relation (5)). This rule depends upon a control parameter, which by analogy with the physical annealing is known as the system temperature.

1.1. Metropolis algorithm and simulated annealing

From statistical mechanic, it is known that at a given temperature T, the probability of finding a system in a state of energy E is given by the Boltzmann distribution

$$\Pr\{E = x\} = k_T \exp(-x/(k_b T)) \tag{4}$$

where $k_T > 0$ is a normalizing constant and k_b is the Boltzmann constant. It can be noticed that at high temperature, the system is more likely to be in a high-energy state than at low temperature. Thus, as T decrease, the range of the Boltzmann distribution concentrates on states with the lowest energy. When T becomes very low, the system "freezes", and provided that the temperature has been lowered sufficiently slowly, this frozen state will be of minimum energy.

It was Metropolis *et al*. (1953) who first elaborated an algorithm based on the Boltzmann distribution for finding the equilibrium configuration of a collection of atoms at a given fixed temperature. The principle of the Metropolis algorithm is as follows. Consider a system in a current state with energy E_0, we generate a new state by random move on the previous configuration and the resulting new energy E_{new} is computed. If $E_{new} < E_0$, then the system remains in this new state and another new state is generated as before. On the contrary, if $E_{new} \geq E_0$ then the probability of remaining in this new state is given by the so called Metropolis criterion

$$\exp\left(-\frac{E_{new} - E_0}{k_b T}\right) \tag{5}$$

If a move is rejected, we try to get another new configuration from the last accepted configuration. After a large number of such iterations, the system eventually reaches a state of equilibrium for the temperature T, and the probability distribution of the accepted configurations satisfy the Boltzmann distribution (4).

For optimization purposes, Kirkpatrick *et al.* (1983) proposed to use Metropolis algorithm (MA) together with an annealing schedule which defines the law of decrease of the temperature (exponential in their case). Starting from a high initial temperature T_{init}, the Metropolis algorithm is applied until a state of equilibrium is reached. The temperature is then lowered in accordance to the annealing schedule, and the Metropolis algorithm is then applied with this new temperature until the obtention of a new equilibrium and so on. This process is repeated until a specified final temperature T_{final} is reached. As we can see, SA is a sequence of MA with a rule of decrease of the temperature from T_{init} to T_{final}. If the decrease of the temperature is sufficiently slow, then the system will reaches a state of minimum energy, corresponding to the global minimum of the cost function.

1.2. Simulated annealing algorithm

Using the principles discussed above, we can solve the optimization problem (3) via the following general simulated annealing algorithm.

1. Choose an initial temperature T_{init} and set the current temperature T to T_{init}: $T = T_{init}$. Select an initial vector parameters q and compute the corresponding cost function $J(q)$.

2. Randomly select a new candidate solution q_{new} in the vicinity of q, and compute the corresponding cost function $J(q_{new})$.

3. Compare $J(q)$ and $J(q_{new})$ using the Metropolis criterion (5) as follows. Let $\Delta J = J(q_{new}) - J(q)$. Accept the new vector parameters q_{new} if $\Delta J < 0$ (i.e. set $q = q_{new}$). In the case where $\Delta J \geq 0$, a number u in $[0, 1]$ is drawn randomly according to a uniform distribution. The new point q_{new} is accepted if $r \leq \exp(\Delta J / T)$, where T is

the current temperature; otherwise it is rejected i.e. q remains unchanged. Equivalently, the new point q_{new} is accepted if it satisfies $J(q_{new}) \leq J(q) - T\log(r)$.

4. Repeat steps 2 and 3 until the sequence of accepted points have reached a state of equilibrium.

5. The temperature T is lowered to a new temperature T_{new} in accordance with the annealing schedule, set $T = T_{new}$ and return to step 2. This process is continued until some stopping rule is satisfied.

There are many way in which this algorithm can be implemented. In what follows, we give some practical rules widely used for an efficient implementation of the simulated annealing algorithm.

Choice of the initial temperature. The initial temperature must be chosen sufficiently large so that any point of the search domain $\mathcal{D}$ has a reasonable chance of being visited. However, if T_{init} is too large then a too long time is spent in a state of "high energy" (i.e. high values of the cost function). Many methods have been proposed in the literature to determine the initial temperature (see for instance Ben-Ameur, 2004). A well accepted approach consist in computing an initial temperature such that the acceptance ratio is approximately equal to a given value τ_0. This can be done as follows. Generate at random η samples uniformly distributed in $\mathcal{D}$: $q^i \in \mathcal{D}$, $i = 1,\ldots,\eta$, and choose a rate of acceptance τ_0, then evaluate the initial temperature using: $T_{init} = -(\Delta J)_{max} / \log(\tau_0)$, where $(\Delta J)_{max}$ is defined as: $(\Delta J)_{max} = \max_{1\leq i\leq\eta} J(q^i) - \min_{1\leq i\leq\eta} J(q^i)$. By the way, we can use the η samples to select an initial decision vector q as follows: $q = \arg\min_{1\leq i\leq\eta} J(q^i)$.

Generation of a new candidate solution (step 2 of the SA algorithm). Generally, a new candidate solution q_{new} is generated by adding a random perturbation to the current solution q. There are many way to do that, a common rule for continuous optimization problem is to add a n_q-dimensional Gaussian random variable to the current value q (see spall): $q_{new} = q + g(\Sigma)$, where g is a zero-mean Gaussian random vector with covariance matrix Σ, which must be fixed by the user. Another approach consists in changing only one component of q at a time (Brooks & Morgan, 1995). This is done by first selecting one of the components of q at random, and then randomly selecting a new value for that variable within its bounds. In Bohachevsky et al 1986, a spherical uniform perturbation is adopted. More precisely, the new candidate point q_{new} is obtained by first generating a random direction vector θ, with $\|\theta\|_2 = 1$, then multiplying it by a fixed step size β, and finally summing the resulting vector to q, i.e. $q_{new} = q + \beta\theta$. The value of the step size β must be set by the user. In a similar way one can also adopt the following rule: $q_{new} = q + CU$, where U is a vector of uniform random number in the range $(-1, 1)$ and C is a constant diagonal matrix whose elements define the maximum change allowed in each component of q. The matrix C is also user defined. The methods presented above are not limitative and some other approaches have been proposed in the literature see for instance Vanderbilt & Louie (1984), Parks (1990) to cite only a few.

Number of repetitions of the Metropolis criterion. At step 4, the repetition of the steps 2-3 is maintained until one of the 2 following conditions is satisfied:

- N_s acceptances
- $N > N_S$ perturbations attempted (i.e. N iterations of the metropolis procedure).

where the integers N and N_s have to be set by the user. In Dreo *et al.* (2006) the following setting are recommended $N_S = 12n_q$ and $N = 100n_q$, where $n_q = \dim(q)$, i.e. n_q is the number of parameters of the problem. Note that this setting is only indicative and some other choices can be adopted by the user.

Annealing schedule. At step 5, the temperature T must be lowered to a new temperature T_{new}. To this end, the geometrical law of decrease: $T_{new} = \lambda T$, with λ constant, is a widely accepted one, because of its simplicity. The constant $\lambda \in (0,1)$ is a user defined parameter which defines the annealing schedule. Usually, λ is set to 0.8 or 0.9. Some other more sophisticated laws of decrease can be used, for a more detailed study see for instance Dreo *et al.* (2006) and references therein.

Stopping rule. It is very difficult, if not impossible, to define a stopping rule which guarantees to stop when the global minimum has been detected, or when there is a sufficiently high probability of having reached it. Consequently, the stopping rules usually adopted all have a heuristic nature. In practice, the SA algorithm is stopped when one of the following conditions is satisfied:

- The final temperature T_f specified by the user (eg. $T_f = 10^{-8}$) has been reached.

- There is no improvement in the solution i.e. the number of consecutive rejections, exceed a given value N_f fixed by the user. Generally, N_f is a multiple of N (i.e. the number of iterations of the metropolis procedure), for instance in our applications, we have chosen: $N_f = 10N$.

Parameter setting in brief. Table 1 summarizes the usually adopted parameters of a simulated annealing algorithm.

Table 1. *Usual parameter setting of a Simulated Annealing.*

Initial temperature (T_{init}) and starting point.	See section "Choice of the initial temperature"
Final temperature	~ 10^{-8}
Rule of decrease of temperature	$T_{new} = \lambda T$, with λ = 0.8 or 0.9
Acceptance rule	Metropolis criterion (see relation (5))
Maximum number of successes within one temperature (N_S)	$N_S = 12 \times \dim(q)$
Maximum number of tries within one temperature (N)	$N = 100 \times \dim(q)$

1.3. Advantages and disadvantages of simulated annealing

The main advantage of the SA is that it achieves a good quality solution, i.e. the absolute error to the global minimum is generally lower than that obtained via other metaheuristics. Moreover, it is versatile and easy to implement. The main drawbacks of SA lie mainly in the choice of the various parameters involved by this algorithm, in particular: the initial temperature, the rule of the decrease of the temperature, the final temperature, the maximum number of consecutive

rejections, the maximum number of tries within one temperature, the maximum number of successes within one temperature. The results obtained are indeed very sensitive to the parameter settings. Consequently, the problem of the selection of the "good parameters" (for a given cost function) is a crucial issue, which is however not yet entirely solved. Another weakness of the method, linked to the problem of parameter setting, is its excessive computing time in most applications. More detailed developments on SA, both practical and theoretical, can be found in Hajec (1988), Ingber (1994), Locatelli (2000), Spall (2003), Dreo *et al.* (2006).

2. Genetic Algorithm

Genetic Algorithms (GA) are a particular class of Evolutionary Algorithms (EA), also known as Evolutionary Computation (EC). The term EC refers to a class of random search methods that are built by analogy with biological evolution, and which implement the principles of the Darwinian theory of *natural selection* and genetics (see table 2). These kinds of methods are fairly simple and general, and thus can be used to solve a wide range of optimization problems. This section focuses on the most widely used method of EC i.e. genetic algorithm.

Table 2. *Analogy between Biological evolution and Genetic Algorithm.*

Biological evolution	**Genetic Algorithm**
Individual	Potential solution to a problem
Chromosome	Encoding of a candidate solution
Population	Set of potential solutions
Generation	Iteration of the algorithm
Crossover, mutation	Search operators
Natural selection	Reuse of good solutions
Fitness	Quality of a solution (i.e. value of the cost function)
Environment	Problem to be solved

Genetic algorithm, is a population-based stochastic search technique introduced by J. H. Holland in 1962 (Holland, 1962; Holland, 1975) and popularized by D. E. Goldberg in 1989. This approach uses a population of points containing several potential solutions, each of which is evaluated and a new population is created from the best of them via randomized operators, such as *selection*, *crossover* and *mutation*, inspired by the natural reproduction and evolution of living creatures. The process is continued through a number of generations (i.e. iterations) with the aim that the population evolves toward an acceptable solution.

A fundamental difference between GA and the simulated annealing (SA, see section 1) is that GA deal with *populations* of potential solutions, or *individuals*, rather than with single solutions. An interesting thing with population-based method is its intrinsic parallelism. This does not mean that solutions evolve independently of each to other, on the contrary, in GA these ones interact, mix together and produce “children” that, hopefully, retains the good characteristics of their parents.

Another peculiarity of GA is that it works not directly on the solution space but utilizes an encoding of the decision variables. Usually, decisions variables are coded as a finite-length string over a finite alphabet. Such a string is called, by analogy to biological evolution, a *chromosome*. The most commonly used representation in GA is the binary alphabet {0, 1} although other representations can be used, e.g. ternary, integer, real-valued etc. For instance, a problem with two variables $q = [q_1 \quad q_2]^T$, may be encoded onto the following binary string:

$$\overbrace{\underbrace{101101001110}_{q_1}:\underbrace{110100101101}_{q_2}}^{q}$$

where q_1 and q_2 are coded with 12 bits, reflecting, presumably, the desired level of accuracy of the decision variables. Examining the chromosome string in itself gives no information about the problem that we want to solve. Conclusions can be drawn only from the decoding of the chromosome i.e. via a come back to the original representation. The decoding step is necessary to asses the performance, or *fitness*, of individuals member of a population. This is done through an objective function, or cost function, that characterizes the performance of an individual i.e. its ability to be a solution of the problem that we are trying to solve. Since the search operates on encoding of decision variables, rather than the decision variables themselves, any GA must incorporate a process of encoding and decoding variables, except obviously, when real-valued representations are adopted (Michalewicz, 1998).

2.1. The main steps of a Genetic Algorithm

As shown Figure 2, a GA starts with an initial population of N randomly chosen individuals in the search space $\mathcal{D}$. The population size N is generally kept constant from generation to generation. At each generation (or iteration) of the GA, the following steps are performed (see also Figure 2).

1. **Evaluation.** The fitness of every individual q of the current population is evaluated. This requires for each q the computation of the cost function $J(q)$.

2. **Selection.** On the basis of their fitness, multiple individuals are randomly selected from the current population.

3. **Reproduction.** The selected individuals are modified using "genetic operators", mainly crossover and mutation, to form a new population, which will be used in the next iteration.

This process is repeated until some stopping rule is satisfied. Usually, the algorithm terminates when either a maximum number of iterations has been reached, or when no further improvement is found.

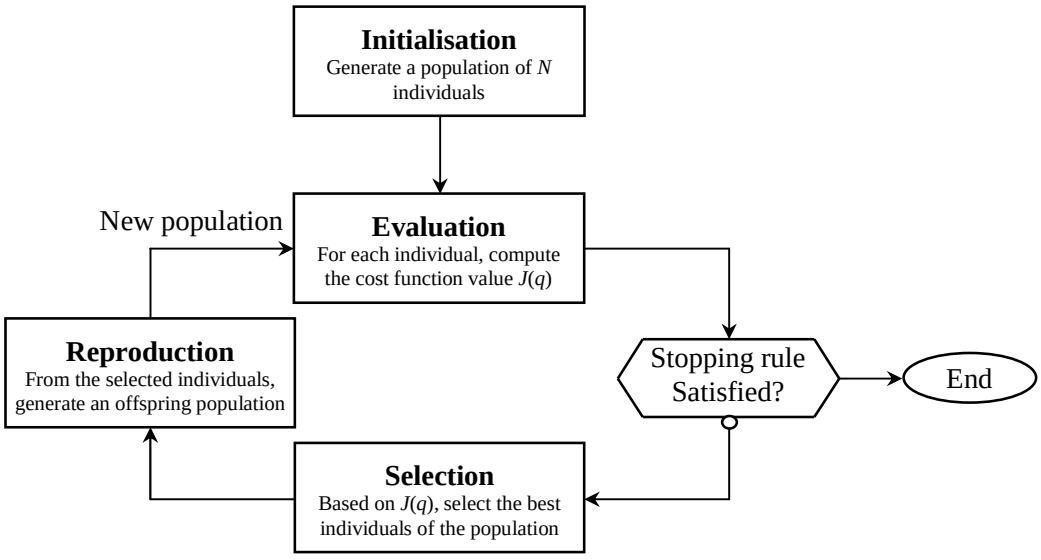

Figure 2 – *Main steps of a GA.*

2.1.1. The initialization step and the representation of the decision variables

Initialization. The initial population of individuals is usually generated at random by sampling the search space according to a uniform distribution. Practically this can be done as follows. Let $\underline{q} = [\underline{q}_1 \ \underline{q}_2 \cdots \underline{q}_{n_q}]^T$ and $\overline{q} = [\overline{q}_1 \ \overline{q}_2 \cdots \overline{q}_{n_q}]^T$ the lower bound and the upper bound, respectively, of the decision vector $q = [q_1 \ q_2 \cdots q_{n_q}]^T$. In other words, the search domain is the hyperbox defined in (3): $\mathcal{D} = \left\{ q \in \mathbf{R}^{n_q} : \underline{q} \leq_e q \leq_e \overline{q} \right\}$. A population of *N* individuals uniformly distributed on $\mathcal{D}$ can then be obtained as follows:

$$q^i = (\overline{q} - \underline{q}) \otimes U + \underline{q}, \quad i = 1, \cdots, N \tag{6}$$

where $\otimes$ stands for componentwise product, q^i is the i^{th} individual of the population and *U* is a n_q-dimensional vector of random numbers uniformly distributed over [0, 1]. This uniform random sampling, ensures covering of the entire range of possible solutions. The population size *N* is a user defined parameter. Sizes of 20-500 individuals are common in practice. In general, *N* should be chosen proportionally to the number of decision variables of the optimization problem, and large enough to allow a good exploration of the search space. There is no rule to fix the optimal population size but there is clear tradeoff between *N* and the computation time.

Coding and decoding the variables. As seen before, an essential aspect of GA is the encoding of the decision variables $q = [q_1 \ q_2 \cdots q_{n_q}]^T$ for performing the GA operations and the associated decoding to return to the original representation. Usually, decisions variables are coded as a string to facilitate the operations performed by the GA. Although this can be done in many ways (see for instance Dréo *et al.*, 2006), we consider here the bit-string representation because of its wide use in practical applications. Usually, the binary coding is performed in such a way that the minimal value of a decision variable[3] q_i is coded by $(0\,0\,0\cdots0)$ and its maximal value is coded by $(111\cdots1)$. This can be done as follows:

$$\widehat{q}_i = \text{rnd}\left[\frac{q_i - \underline{q}_i}{r_i} \right], \text{ with: } r_i = \frac{\overline{q}_i - \underline{q}_i}{2^b - 1}, \quad i = 1, \cdots, n_q \tag{7}$$

where the operator rnd(.) rounds off the argument to the nearest integer, *b* is the number of bit desired for the binary representation and r_i is the resolution (or precision) of the coding with *b* bits. The integer $\widehat{q}_i$ is then represented using the standard binary representation $(a_1^i \ a_2^i \ a_3^i \cdots a_b^i)$:

$$\widehat{q}_i = \sum_{j=1}^{b} a_j^i \, 2^{b-j}, \quad i = 1, \cdots, n_q \tag{8}$$

where the elements a_j^i are ether 0 or 1. The complete coding of *q* is then obtained by the concatenation of the $(a_1^i \ a_2^i \ a_3^i \cdots a_b^i)$:

$$\overbrace{(\underbrace{a_1^1 \ a_2^1 \ a_3^1 \cdots a_b^1}_{\text{Coding of } q_1} \quad \underbrace{a_1^2 \ a_2^2 \ a_3^2 \cdots a_b^2}_{\text{Coding of } q_2} \quad \cdots \quad \underbrace{a_1^{n_q} \ a_2^{n_q} \ a_3^{n_q} \cdots a_b^{n_q}}_{\text{Coding of } q_{n_q}})}^{\text{Coding of } q} \tag{9}$$

[3] Recall that q_i is the i^{th} component of the vector *q*.

Inversely, the decoding of a *b*-bits representation $(a_1^i\ a_2^i\ a_3^i \cdots a_b^i)$ is given by:

$$q_i = \underline{q}_i + \frac{\overline{q}_i - \underline{q}_i}{2^b - 1} \sum_{j=1}^{b} a_j^i 2^{b-j}, \quad i = 1, \cdots, n_q \tag{10}$$

2.1.2. The evaluation step and the fitness function

Each individual of the current population is evaluated through a so called fitness function $F(q)$. By definition, better solutions have higher fitness. In the case of a maximization problem, the fitness function is identical to the criterion $J(q)$: $F(q) = J(q)$. For a minimization problem, the best individuals are those which render the cost function as small as possible; in this case the fitness function is the inverse of the criterion $J(q)$: $F(q) = 1/J(q)$. Note that the computation of the fitness value requires the original representation of the decisions variables. This is done using the decoding rule (10) on each individual coded as a bit-string.

2.1.3. The selection step and its operators

At each iteration, *N* individuals of the current population are selected to generate a *parents population*. The selection of the individuals from the actual population is done via a fitness-based process, where betters solutions are more likely to be selected. Usually, the selection method is designed so that a small proportion of less fit solutions are also selected. This helps to preserve the diversity in the population which is required to maintain the exploration of the search domain and thus prevents a premature convergence on poor solutions. The most widely used selection operators are roulette wheel selection and tournament selection.

Roulette wheel selection. The individuals are drawn at random with replacement from the current population with a probability that increase with their fitness. To this end, a real valued interval [0, Σ] is determined, where Σ is the sum of the individuals fitness in the current population:

$$\Sigma = \sum_{i=1}^{N} F(q^i) \tag{11}$$

The individuals are then mapped to contiguous segments in the range [0, Σ], such that each individual segment is equal in size to its fitness. For instance, in Figure 3 the length of the line is the sum of the seven individual's fitness. The individual 7 has the largest fitness value and occupies the largest segment whereas the individuals 4 and 6 are the least fit and have correspondingly smaller segments within the line.

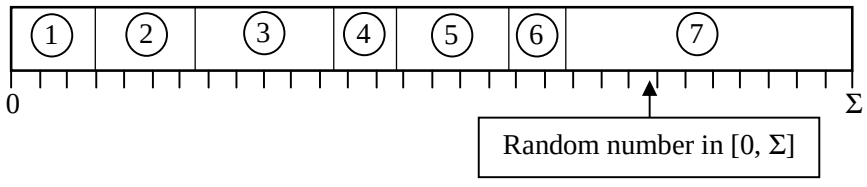

Figure 3 – *Roulette wheel selection.*

The selection is performed by generating a random number, uniformly distributed in the interval [0, Σ]; the individual whose segment spans the random number is selected. This process is repeated until the desired numbers of individuals is obtained. This method is similar to a roulette wheel with each slice proportional in size to the fitness.

Tournament selection. A number *N'* of individuals is chosen randomly from the current population and the best individual from this group is selected as parent. This process is repeated until the desired numbers of individuals is obtained. The parameter for tournament selection is the tournament size *N'*. This parameter takes values from 2 to *N* (i.e. the population size). A detailed study about the tournament selection can be found in Miller & Goldberg (1995).

2.1.4. The reproduction step and its operators

A repeated selection from the same population produces nothing more than copies of the individuals originally in it with a preference for the best ones. To hope an improvement, some variations in the parents population must be done. The aim of the reproduction step is to produce a new population from the parents that were selected from the current population. To this end, two kinds of operator can be used: crossover operator (or recombination operator) and mutation operator.

Crossover operator. Pairs of parents are combined to form via crossover operation two new individuals (the childs) that inherit many characteristics of their parents. There are lots of possibilities for defining such an operator, depending on the problem and its encoding. In the case of a coding by bit-strings, the simplest form is the so called one-point crossover. For each pair of parents, the one-point crossover is performed with a given probability p_c. If crossover occurs, an integer k is generated at random according to a uniform distribution between 1 and $n_q \times b$-1, and the last $n_q \times b$-k bits of each parent are exchanged to produce two childs, as illustrated Figure 4.

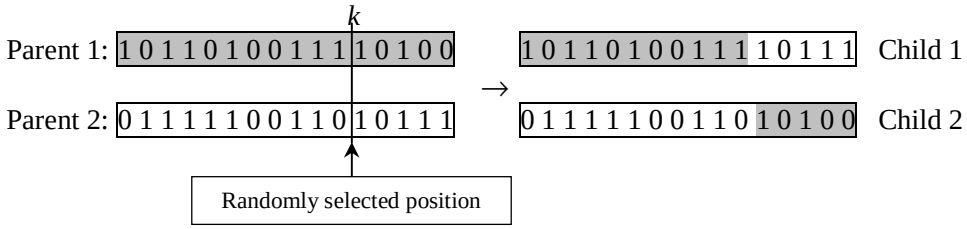

Figure 4 – *One-point crossover operator.*

The crossover probability p_c is a user defined parameter. Usually, p_c is chosen in the interval [0.5, 0.95], this mean that for a selected pair of parents there is a chance of crossover between 50 and 95%. If the crossover operation is not performed, the two childs are identical to their parents. The crossover operation can be generalized for more than one-point crossover (see Dréo *et al.*, 2006).

Mutation operator. The main objective of mutation is to provide new individuals that cannot be generated otherwise. This step is essential because it allows the exploration of new regions where, perhaps, good solutions can be found. This is in contrast with the selection and crossover operators which focus attention on promising region of the search space. From this point of view, selection and crossover operators permit the exploitation of promising regions, whereas mutation operator allows the exploration of new regions. These two aspects, exploration and exploitation, are essential to increase the probability of finding a global optimal solution.

Each individual of the population obtained via crossover is submitted to mutation with a given probability p_m. The simplest mutation operation consist of choosing at random a position between 1 and $n \times b$, and substituting the character in that position by another character of the alphabet. For instance, in the case of a bit-string representation, the mutation is obtained simply by flipping a randomly chosen bit (see Figure 5).

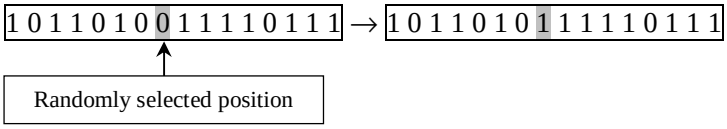

Figure 5 – *Mutation operator.*

The mutation probability p_m is also a user defined parameter. This probability is usually chosen much lower than the crossover probability to give preference to the exploitation phase. Value between 0.001 and 0.07 are common in practice.

2.1.5. The stopping rule

After the reproduction step, a new population of N individuals is obtained, which will be used in the next iteration of the algorithm (see figure 2). The various operations detailed above: evaluation, selection and reproduction are then repeated until a termination condition has been reached. There has been very few theoretical studies about when to stop a GA (Eiben & Schoener, 2002). The usual stopping criterion is a fixed number of iterations, but this does not guarantee the convergence of the algorithm to a solution. A more satisfying stopping rule consist in detecting that no significant improvement was found during a certain number of iterations N_G. In this way, we can stop the GA when the following condition is satisfied:

$$\left| \frac{1}{N_G} \sum_{j=1}^{N_G} F(q_{best}(i-j)) - \frac{1}{N_G} \sum_{j=1}^{N_G} F(q_{best}(i-j-N_G)) \right| \leq \rho_{GA} \qquad (12)$$

where ρ_{GA} is the minimum level of improvement desired on N_G iterations, $q_{best}(l)$ is the best individual at generation l and i is a multiple of N_G. Possible values of ρ_{GA} and N_G are, respectively: 0.001 and 10.

2.2. The standard genetic algorithm

Although there are many variations in implementing the GA, we present here a fairly standard form of this algorithm.

1. **(Initialisation).** Randomly generate an initial population of N individuals: $q^1, q^2, \cdots, q^N$ (with N even) and evaluate the fitness function $F(q^i),\ i = 1, \cdots, N$. Each individual q^i (a point in search space) is encoded, for instance, into a bit-string.

2. **(Parent selection).** Select with replacement N parents from the current population, and group them randomly in pairs. The parents are selected according to their fitness; the individuals having higher fitness value being selected more often.

3. **(Reproduction: crossover).** For each pair of parents, resulting of step 1, generate a random number r uniformly in the range [0, 1]. If $r \leq p_c$, then generate a uniform random integer k in the range [1, $n_q \times b$-1] and exchange the $n_q \times b$-k elements of each parent to the right of element k. If no crossover take place (i.e. $r > p_c$), then form two offsprings that are exact copies of the parents.

4. **(Reproduction: mutation).** For each individual resulting of step 3, generate a random number r uniformly in the range [0, 1]. If $r \leq p_m$, then generate a uniform random integer k in the range [1, $n_q \times b$], and switch the element k of the bit-string from 0 to 1 or vice versa.

5. **(Evaluation and stopping rule).** Via a decoding rule, compute the fitness value of each individual of the new population resulting of step 4. Terminate the algorithm if the stopping rule is satisfied (i.e. a maximum number of iteration has been reached or no further improvement is found); else go to 2.

As it is usual with stochastic algorithms, there are many choices to do for a practical implementation of the GA (see Spall, 2003). In particular: the encoding rule, the population size (N), the probability distribution generating the initial population, the strategy for parent-selection, the number of points-crossover, the crossover probability (p_c) and the mutation probability (p_m), the stopping rule. Table 3 summarizes the choices most widely adopted in practical applications. A more detailed study on this issue can be found in Lobo, Lima & Michalewicz (2007).

Table 3. *Parameters of a standard Genetic Algorithm.*

Encoding rule	Bit-string of length $n_q \times b$, where $n_q = \dim(q)$
Population size (N)	$20 \leq N \leq 500$
Probability distribution	Uniform on the search space
Parent-selection	Roulette wheel
Number of points crossover	One-point crossover
Crossover probability	$0.5 \leq p_c \leq 0.95$
Mutation probability	$0.001 \leq p_m \leq 0.07$

2.3. Advantages and disadvantages of GA

The main advantage of GA (and its many versions) is its robustness as well as its intuitiveness, ease of implementation, and the ability to deal successfully with a wide range of difficult problems. By robustness it must be understood that, within fairly wide margins, the problem of adjusting the parameters is not very critical. This insensitivity makes it possible to find acceptable solutions without excessive effort. A main drawback with GA is that some well adapted individuals (compared to the other members of the population, but faraway from the optimum point), dominate the population, causing it to converge on a local minimum. In these conditions, the probability of finding better solutions is very small because crossover between similar individuals, produces little changes. Only mutation remains to seek the best individuals, but this is generally not sufficient for a fast convergence toward the best solution. The latter requires thus an excessive computational time.

3. Particle Swarm Optimization (PSO)

Swarm intelligence (SI) can be defined as the apparent intelligence that is emergent from the collective behavior of decentralized systems. In other words, a certain type of intelligence can emerge from the perpetual interaction of self-organized entities. SI systems are generally composed of a population of simple agents, interacting locally with each other and with their environment. The main feature of these systems is that they do not have a centralized control which imposes the behavior of each agent. Instead, each agent is governed by its own rules and the local interactions between them lead to the emergence of an intelligent global behaviour, unknown to the individual agents[4]. Due to this interesting feature, optimization techniques inspired by swarm intelligence have received a lot of attention during the past years, and various techniques have been proposed in the literature. Among them, the most widely used swarm intelligence algorithm in the context of continuous optimization is indubitably the Particle Swarm Optimization (PSO).

PSO is a relatively recent stochastic optimization technique developed by J. Kennedy and R. Eberhart in 1995. GA and PSO are similar in the sense that these two approaches are population-based random search methods but with different strategies of evolution. PSO draws its inspiration from the collective behavior of living beings, including the notion of collective intelligence of a population of individuals (Kennedy & Eberhart, 1995). It is a population based search algorithm where each individual is called *particle* and represents a candidate solution. Each particle evolves through the search space seeking the optimal solution of the optimization problem. A particle *i* of a swarm is characterized by its position q^i and its change in position v^i, called *velocity*. For seeking the optimal solution, each particle utilizes two kind of information: the memory of its own best position and the knowledge of the global best position found by the group[5]. The movement of a particle is then adjusted according to its velocity and the difference between its current position, the best position found by the group and the best position it has found so far. The repetition of this procedure leads the swarm toward a domain of the search space containing, hopefully, high-quality solutions.

This section on PSO is organizes as follows. In section 3.1, the dynamic of the particle swarm is described. Notably, the updating rule of position and change in position are presented with some details. Section 3.2 presents a general algorithm for a practical implementation of PSO, and the standard values of the user defined parameters are given. Finally, section 3.3 presents the majors advantages and inconvenient of PSO.

3.1. Dynamic of the particles of a swarm

Consider a swarm of *N* particles. The position of a particle *i* ($i = 1, \cdots, N$), is denoted q^i, where $q^i = [q_1^i \;\; q_2^i \cdots q_{n_q}^i]^T$ is the n_q-dimensional vector of decision variables of the optimization problem (3). The change in position or velocity of a particle *i* is denoted v^i, where $v^i = [v_1^i \;\; v_2^i \cdots v_{n_q}^i]^T$ is the n_q-dimensional vector of change in decision variables. The change in position of a particle *i* at iteration *k*+1 is defined by: $v^i(k+1) = q^i(k+1) - q^i(k)$; the movement of a particle is then governed by the equation:

$$q^i(k+1) = q^i(k) + v^i(k+1) \tag{13}$$

[4] In Nature, this kind of behavior can be observed in colonies of insects (e.g. ants), flocks of birds or schools of fish.

[5] In a minimization problem, the term "best" must be understood as the position with the smallest objective value.

The key point lies in the manner in which the velocity is modified over time. The updating rule of velocity is done in such a way that the swarm of particles mimics the collective behaviour observed on living beings. According to the observation of Boyd and Richardson (Boyd & Richardson, 1985), human beings utilize two important kinds of information in the decision process. The first one is their own experience, i.e. they have tried the choice and know which state has been better so far and also how good it was. The second one is the experience of others, i.e. the knowledge about how the other agents around them have performed. By analogy with these observations made about social behaviors, the velocity of a particle *i* is modified according to its own previous best solution p^i and its group's previous best solution p_g, with the aim to get an improvement (i.e. in the sense of a decrease of the cost function). Hence, the updating rule of the velocity of the particles is dependent on their current speed and position, the best preceding position p^i (i.e. corresponding to the lowest cost function *J*) and the best position p_g of the group: $v^i(k+1) = f(q^i(k), v^i(k), p^i, p_g)$. The function of evolution *f* allowing updating the velocity of a particle *i*, is usually implemented as follows:

$$v^i(k+1) = w(k)v^i(k) + \varphi_1 \otimes (p^i - q^i(k)) + \varphi_2 \otimes (p_g - q^i(k)) \tag{14}$$

where $w(k)$ is known as the inertia weight; this factor is used to reduce the growth in velocity of the particle, more details on $w(k)$ will be provided below. In the relation (14), φ_1 and φ_2 are two random functions defined as:

$$\varphi_1 = c_1 U_1, \quad \varphi_2 = c_2 U_2 \tag{15}$$

Where U_1 and U_2 are two random vectors whose components are uniformly generated in the range (0, 1), c_1 and c_2 are two positive constants known, respectively, as the *individual* (or *cognitive*) *coefficient* and *social coefficient*. These user defined coefficients are usually set about 2. The symbol $\otimes$ stands for a componentwise vector multiplication. The updating rule (14) shows that the velocity of a particle is determined by its velocity $v^i(k)$ and the so called *individual* and *social* parts. The individual part $\varphi_1 \otimes (p^i - q^i(k))$, represents the tendency of the particle to return to the best position it has visited so far whereas the social part $\varphi_2 \otimes (p_g - q^i(k))$ represents the tendency of the particle to be attracted towards the best position found by the swarm (Blum & Li, 2008). Figure 6 gives a geometrical interpretation of the updating rule (14).

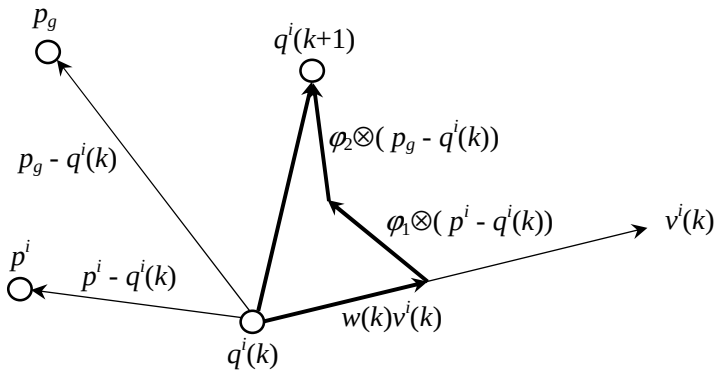

Figure 6 – *Geometrical interpretation of the updating rule.*

The inertia weight. The inertia coefficient $w(k) \in [w_{min},\ w_{max}]$ is used to control the growth of the velocity of the particles i.e. the stability of the swarm. To this end, $|w(k)|$ must be lower than one. Note that a positive inertia coefficient introduces a preference for the particle to continue moving in the same direction it was going previously. A progressive decreasing value of w over time introduces a progressive transition from exploratory (global search) to exploitative (local search) mode. This decrease occurs between the bounds w_{max} and w_{min} defined by the user. Usually $w(k)$ is reduced linearly according to the following rule:

$$w(k) = \frac{(\texttt{MaxIter} - k)(w_{max} - w_{min})}{\texttt{MaxIter}} + w_{min} \qquad (16)$$

where `MaxIter` is the maximum number of allowed iterations. Typically, w_{min} is set about 0.3 or 0.4 and w_{max} is set about 0.8 or 0.9.

PSO with constriction coefficient. A variant of the updating rule (14), proposed by Clerc (Clerc & Kennedy, 2002), uses what is called a *constriction factor* noted χ. With this variant, the update of the velocity is done as follows:

$$\begin{aligned} & v^i(k+1) = \chi[v^i(k) + \varphi_1 \otimes (p^i - q^i(k)) + \varphi_2 \otimes (p_g - q^i(k))] \\ & \varphi_1 = c_1 U_1, \quad \varphi_2 = c_2 U_2 \\ & \chi = \frac{2\kappa}{\left|2 - \varphi - \sqrt{\varphi^2 - 4\varphi}\right|}, \text{ with : } \varphi = c_1 + c_2, \text{ and } \varphi > 4 \end{aligned} \qquad (17)$$

Note that the constriction factor corresponds to the use of a constant inertia weight: $w(k) = \chi$ for all k. This approach was introduced to ensure the stability of the swarm i.e. to avoid the divergence of the particles beyond the boundaries of the search space (for more detail see Clerc & Kenndy, 2002). With this variant, the parameter setting usually adopted is: $\kappa = 1,\ c_1 = c_2 = 2.05$, thus $\varphi = 4.1$ and $\chi \approx 0.73$.

3.2. The standard PSO algorithm

According to the principles discussed above, we can solve the optimization problem (3) using the following algorithm.

1. **(Initialisation).** Set $k = 0$. Generate an initial population of N particles: $q^1(0), q^2(0), \cdots, q^N(0)$. Generate the corresponding initial velocities: $v^1(0), v^2(0), \cdots, v^N(0)$. Set the local bests p^i's to $p^i = q^i(0),\ i = 1, \cdots, N$. Set the global best p_g to $p_g = \arg\min_{1 \le i \le N} J(q^i(0))$.

2. **(Swarm evolution).** Set $k = k + 1$. Update the velocity of each particle according to: $v^i(k) = f(q^i(k-1), v^i(k-1), p^i, p_g),\ i = 1, \cdots, N$, where f is given by (14) or (17). Update the position of each particle using: $q^i(k) = q^i(k-1) + v^i(k),\ i = 1, \cdots, N$.

3. **(Update the p_i's and p_g).** For each particle, update the local best position found so far using the following rule: If $J(q^i(k)) < J(p^i)$ then $p^i = q^i(k),\ i = 1, \cdots, N$. Update the global best position as follows: $p_g = \arg\min_{1 \le i \le N} J(p^i)$.

4. **(Stopping rule).** Terminate the algorithm if the stopping rule is satisfied (i.e. a maximum number of iteration has been reached or no further improvement is found); else go to 2.

Initialization. The initial population of particles is usually generated at random by sampling the search space according to a uniform distribution. This can be done in a similar way as in GA, see section 2.1.1. The initial velocities can also be generated at random but this requires defining a sample space. To this end we can introduce a parameter bound v_{max}. The initial velocities can then be obtained by sampling the space $\mathcal{V} = \left\{ v \in \mathbf{R}^{n_q} : -\mathbf{1}v_{max} \le_e v \le_e \mathbf{1}v_{max} \right\}$ according to a uniform distribution. The symbol **1** represents the unit vector and $\le_e$ indicates element-by-element inequality. The swarm size N is a user defined parameter. Sizes of 20-150 particles are common in practice. Concerning this parameter, the same things as in GA apply (see section 2.1.1).

Stopping rule. A simple stopping criterion is to fix a maximum allowed number of iterations `MaxIter`. This parameter is required to use a varying inertia weight (see relation (16)). However, the use of this criterion alone cannot guarantee the convergence of the algorithm to a solution. Instead, it is preferable to stop the algorithm when no improvement is found after a given number of iteration N_I. To this end we can use the same stopping rule as in GA, but in our experiments, we have found better to stop the algorithm when the following condition is satisfied:

The algorithm stops when: $\left| J(p_g(k)) - J(p_g(k - N_I)) \right| \le \rho_{PSO}$ (18)

where k is the current iteration, ρ_{PSO} is the minimum level of improvement desired on N_I iterations, and $p_g(l)$ is the global best found at iteration l. Possible values of ρ_{PSO} and N_I are, respectively: 0.001 and 50.

Parameter of PSO algorithm. As in GA, there are many choices to do for a practical implementation of the PSO algorithm. In particular: the swarm size (N), the probability distributions generating the initial positions and the initial velocities, the maximal allowed velocity, the bound of the inertia weight and its rule of decrease, the cognitive factor and the social factor. Table 4 summarizes the choices most widely adopted in practical applications. The values given in this table are only indicatives and some variations can be required to improve the performance. But this of course is not at all easy to do particularly when the number of parameters is important.

Table 4. *Parameters of a standard PSO Algorithm.*

Swarm size (N)	$20 \le N \le 150$
Initial positions	randomly (uniformly) generated in the search space
Maximal velocity (v_{max})	v_{max} can be set to the maximal bound of q
Initial velocities	randomly (uniformly) generated in $[-v_{max}, v_{max}]^n$
Bounds of the inertia weight (w_{min}, w_{max})	$0.3 \le w_{min} \le 0.4$, $0.8 \le w_{max} \le 0.9$
Rule of decrease of the inertia weight	Linear from w_{max} to w_{min}
Cognitive factor (c_1)	$c_1 \approx 2$
Social factor (c_2)	$c_2 \approx 2$

3.3. Advantages and disadvantages of PSO

The main advantage of the PSO is its ease of implementation as well as its ability to find good solutions much faster than other metaheuristics (less function evaluations). However, it cannot improve the quality of the solutions as the number of iterations is increased (Angeline, 1998). Similar to the GA, an important drawback with PSO, is that the swarm may prematurely converge. This is mainly because particles converge to a point which is on the line between the global best point p_g and the personal best positions p_i. However this point is not guaranteed to be even a local optimum. Another drawback, similar to the SA, is the great sensitivity of PSO to parameter settings: a small change in parameters may result in a proportionally large effect (Lovberg & Krink, 2002).

4. Heuristic Kalman Algorithm (HKA)

In this section, we introduce a recently developed optimization method called Heuristic Kalman Algorithm (HKA) (Toscano & Lyonnet, 2009). As GA and PSO, HKA falls into the category of the so called "population based stochastic optimization technique". However, its principle is entirely different to other known stochastic algorithms. Indeed, HKA considers the optimization problem as kind of learning process intended to give an estimate of the optimum. It utilizes a Gaussian probability density function (GPDF), a measurement process (MP) and a Kalman estimator (KE) allowing to improve the quality of the estimate obtained through the MP. The GPDF evolves in the search space seeking the optimal solution of the optimization problem. A GPDF is characterized by its mean vector m and its variance matrix Σ. For seeking the optimal solution, the parameters of the GPDF are updated by taking into account sample points obtained through a measurement process; this is done using a Kalman estimator. Indeed, a Kalman estimator can be seen as a mechanism able to update our knowledge about unknown quantities of interest, by taking into account new gained information. The "movement" of the GPDF is then adjusted according to its current mean value and the new information obtained via the measurement process. The repetition of this procedure leads the GPDF toward a domain of the search space containing, hopefully, high-quality solutions.

This section on HKA is organizes as follows. In section 4.1, the principle of the heuristic Kalman algorithm is described. Notably, the updating rules of the parameters of the GPDF are presented with some details. Section 4.2 presents a general algorithm for a practical implementation of HKA as well as the standard values of the user defined parameters. Finally, section 4.3 presents the majors advantages and inconvenient of HKA.

4.1 Principle of the algorithm

The principle of the algorithm is shown Figure 7. The proposed procedure is iterative, and we denote by k, the k^{th} iteration of the algorithm. We have a random generator of probability density function (pdf) $g(q)$, which produces, at each iteration a collection of N vectors that are distributed about a given mean vector $m(k)$ with a given variance-covariance matrix $\Sigma(k)$. This collection can be written as follows:

$$\mathbf{q}(k) = \left\{ q^1(k), q^2(k), \cdots, q^N(k) \right\} \tag{19}$$

where $q^i(k)$ is the i^{th} vector generated at the iteration number k: $q^i(k) = [q_1^i(k), \cdots, q_{n_q}^i(k)]^T$, and $q_l^i(k)$ is the l^{th} component of $q^i(k)$ ($l = 1, \ldots, n_q$).

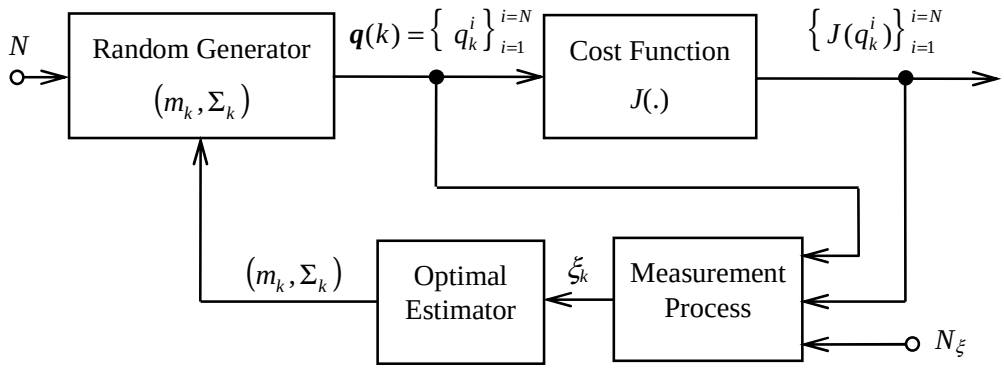

Figure 7 – *Principle of the algorithm.*

This random generator is applied to the cost function *J*. Without loss of generality, we assume that the vectors are ordered by their increasing cost function i.e.:

$$J(q^1(k)) < J(q^2(k)) < \cdots < J(q^N(k)) \tag{20}$$

The principle of the algorithm is to modify the mean vector and the variance matrix of the random generator until a high quality solution is reached. More precisely, let N_ξ be the number of considered best samples, that is such that $J(q^{N_\xi}(k)) < J(q^i(k))$ for all $i > N_\xi$. Note that the best samples are those of the sequence (19) which have the smallest cost function. The objective is then to generate, from the best samples, a new random distribution that approaches the minimum of the cost function *J*. The problem is how to modify the parameters of the random generator to achieve a reliable estimate of the optimum.

To solve this problem, we introduce a measurement procedure followed by an optimal estimator of the parameters of the random generator. The measurement process consists in computing the average of the candidates that are the more representative of the optimum. For the iteration *k*, the measurement, denoted $\xi(k)$, is then defined as follows:

$$\xi(k) = \frac{1}{N_\xi} \sum_{i=1}^{N_\xi} q^i(k) \tag{21}$$

where N_ξ is the number of considered candidates. We can consider that this measure gives a perturbed knowledge about the optimum, i.e.

$$\xi(k) = q_{opt} + v(k) \tag{22}$$

where $v(k)$ is an unknown disturbance, which is centered on q_{opt}, and acting on the measurement process. Note that $v(k)$ is the random vector between the measure $\xi(k)$ and the unknown optimum q_{opt}. In other words, $v(k)$ is a kind of measure of our ignorance about q_{opt}. Of course, this uncertainty cannot be measured but only estimated by taking into account all available knowledge. In our case, the uncertainty of the measure is closely related to the dispersion of the best samples $q^i(k)$ ($i = 1, \ldots, N_\xi$).

Our ignorance about the optimum can thus be taken into account by using the variance vector associated to these best samples:

$$V(k) = \frac{1}{N_\xi}\left[\sum_{i=1}^{N_\xi}\left(q_1^i(k) - \xi_1(k)\right), \cdots, \sum_{i=1}^{N_\xi}\left(q_{n_q}^i(k) - \xi_{n_q}(k)\right)\right]^T \tag{23}$$

In these conditions, the Kalman estimator can then be used to make an estimate, so-called "*a posteriori*", of the optimum, i.e. taking into account the measure as well as the confidence we place in it. As seen, this confidence can be quantified by the variance vector (23).

Roughly speaking, a Kalman filter is an optimal recursive data processing algorithm (Maybeck, 1979). The optimality must be understood as the best estimate which we can make according to the model used for the measurement process as well as the data used to compute this estimate.

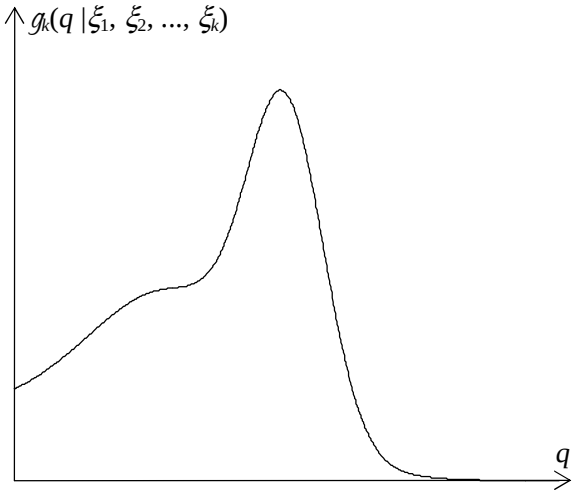

Figure 8 – *Conditional pdf.*

To understand how a Kalman filter works, consider the Figure 8, which depicts a conditional probability density of the value of a scalar quantity q obtained at iteration k, conditioned on knowledge that its measurement at the first iteration is ξ_1 and similarly for iterations 2 through k. This conditional probability density function is denoted as $g_k(q \mid \xi_1, \xi_2, ..., \xi_k)$. In our problem, q is the decision variable related to a given optimization problem, and $\xi_1, \xi_2, ..., \xi_k$ are the successive measurement about the optimal value q_{opt}. Such a conditional probability density contains all the available information about q_{opt}, it indicates, for the given value of all measurements taken up through iteration k, what the probability is that q_{opt} belongs to any particular range of values. The shape of the pdf reflects the amount of uncertainty we have in the knowledge of the value of q_{opt}. If the pdf is a narrow peak, then the most probable values are concentrated in a narrow band of q values. On the contrary, if the pdf is very flat, then the most probable values are spread over large range of q, indicating a large uncertainty about the knowledge of q_{opt}.

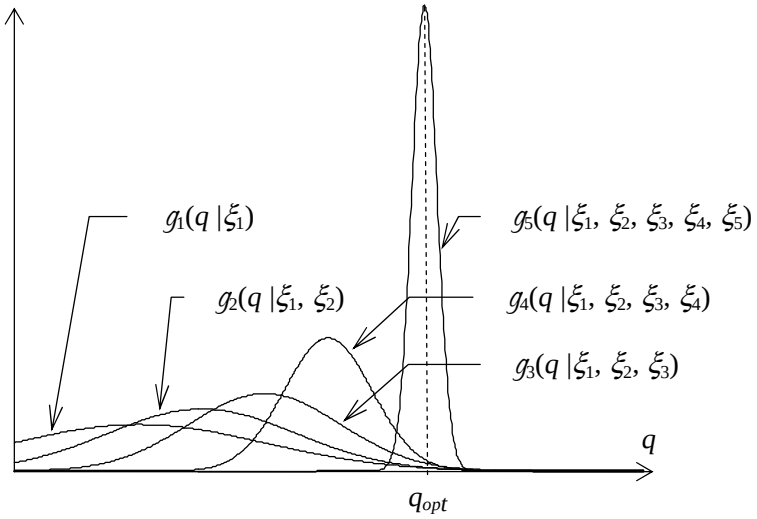

Figure 9 – *propagation of the pdf through the use of the Kalman filter.*

The Kalman filter allows updating the pdf, measurement after measurement (cf. Figure 9); this is what we call the propagation of the pdf. In this way, we see that we start in a state of quasi "complete ignorance" about q_{opt}; but, as we accumulate information through the measurement process, we acquire more and more accurate estimates of the optimum. Note that this corresponds very nicely to the common learning process.

Updating rules of the Gaussian generator. Our objective is to design an optimal estimator that combines a prior estimation of q_{opt} and the measurement $\xi(k)$, so that the resulting posterior estimate is better in the sense of a diminution of the cost function (minimization problem). Based on the Kalman equations, the updating rule of the Gaussian generator are as follows (see Toscano & Lyonnet, (2009) for a detailed derivation):

$$\begin{cases} m(k+1) = m(k) + L(k)(\xi(k) - m(k)) \\ \Sigma(k+1) = (I - a(k)L(k))\Sigma(k) \end{cases} \tag{24}$$

With:

$$L(k) = \Sigma(k)(\Sigma(k) + D(k))^{-1}, \text{ and: } a(k) = \frac{\alpha \min\left(1, \left(\frac{1}{n_q}\sum_{i=1}^{n_q}\sqrt{v_i(k)}\right)^2\right)}{\min\left(1, \left(\frac{1}{n_q}\sum_{i=1}^{n_q}\sqrt{v_i(k)}\right)^2\right) + \max_{1\le i\le n_q}(v_i(k))} \tag{25}$$

where $D(k)$ is a diagonal matrix having in its diagonal the variance vector $V(k)$, $v_i(k)$ represents the i^{th} component of the variance vector $V(k)$ defined in (23), and $\alpha \in (0,1]$ is given by the user (usually a is set about 0.4 to 0.7). The coefficient $a(k)$ is used to control the decrease over time of the variance matrix $\Sigma(k)$. This decrease ensures a progressive transition from global search to local search.

Note that all the matrices used in this formulation (i.e. $L(k)$, $\Sigma(k)$, $D(k)$) are diagonals. Consequently, to save computation time we have to use a vectorial form for computing the various quantities of interest. The vectorial form of (24) and, (25) are given by:

$$\begin{aligned} &m(k+1) = m(k) + \mathrm{diag}(L(k)) \otimes (\xi(k) - m(k)) \\ &\mathrm{diag}(\Sigma(k+1)) = \mathrm{diag}(\Sigma(k)) - a(k)\mathrm{diag}(L(k)) \otimes \mathrm{diag}(\Sigma(k)) \\ &\mathrm{diag}(L(k)) = \mathrm{diag}(\Sigma(k)) \,//\, (\mathrm{diag}(\Sigma(k)) + V(k)) \end{aligned} \tag{26}$$

where the symbol $\otimes$ stand for a element-by-element product and, similarly, // means a element-by-element divide.

4.2. Algorithm

According to the principles discussed above, the minimization of the objective function $J(q)$ (see relation (3)) can be done according to the following algorithm.

1. **(Initialisation).** Choose N, N_ξ and α. Set $k = 0$, $m(k) = m_0$, $\Sigma(k) = \Sigma_0$.

2. **(Gaussian generator).** Generate a sequence of N vectors $q^1(k), q^2(k), \cdots, q^N(k)$, according to a Gaussian distribution parametrized by $m(k)$ and $\Sigma(k)$.

3. **(Measurement process).** Using relations (21) and (23) compute $\xi(k)$ and $V(k)$.

4. **(Updating rules of the Gaussian generator).** Using relations (26) update the parameter of the Gaussian generator.

5. **(Stopping rule).** If the stopping rule is not satisfied go to step 2 otherwise stop. to 2.

The practical implementation of this algorithm requires: an appropriate initialization of the the Gaussian distribution i.e. m_0 and Σ_0; the selection of the user defined parameters namely i.e.: N, N_ξ and α.; the introduction of a stopping rule. These various aspects are considered hereafter.

Initialization and parameter settings. The initial parameters of the Gaussian generator are selected to cover the entire search space. To this end, the following rule can be used:

$$m_0 = \begin{bmatrix} \mu_1 \\ \vdots \\ \mu_{n_q} \end{bmatrix}, \quad \Sigma_0 = \begin{bmatrix} \sigma_1 & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & \sigma_{n_q} \end{bmatrix}, \quad \text{with : } \begin{cases} \mu_i = \dfrac{\overline{q}_i + \underline{q}_i}{2} \\ \sigma_i = \dfrac{\overline{q}_i - \underline{q}_i}{6} \end{cases}, \quad i = 1, \cdots, n_q \tag{27}$$

where $\overline{q}_i$ (respectively $\underline{q}_i$) is the i^{th} upper bound (respectively lower bound) of the hyperbox search domain. With this rule, 99% of the samples are generated in the intervals: $\mu_i \pm 3\sigma_i$, $i = 1, \cdots, n_q$.

We have to set the three following parameters: the number of points N, the number of best candidates N_ξ and the coefficient α. To facilitate this task, table 3 summarizes the standard parameter setting of HKA.

Table 5. *Standard parameter setting of HKA.*

Number of sample points (N)	$20 \le N \le 150$
Number of best candidates	$2 \le N_\xi < N$
Coefficient α	0.4 to 0.9

Stopping rule. The algorithm stops when a given number of iterations `MaxIter` is reached (`MaxIter` = 300 in all our experiments) or a given accuracy indicator is obtained. The latest take into account the dispersion of the N_ξ best points. To this end, we consider that no significant improvement can be done when the N_ξ best points are in a ball of a given radius ρ_{HKA} (e.g. $\rho_{HKA} = 0.005$). More precisely, the algorithm stops when:

$$\max_{2 \le i \le N_\xi} \left\| q^1 - q^i \right\|_2 \le \rho_{HKA} \tag{28}$$

where $\left\| \cdot \right\|_2$ represents the Euclidean norm of its argument, and $q^1, \cdots, q^{N_\xi}$ are the N_ξ best candidate solutions.

In conclusion, the search procedure HKA is articulated around three main components, the Gaussian pdf function $g_k(q)$ (parametrized by $m(k)$ and $\Sigma(k)$, the measurement process and the Kalman estimator. Sampling from the pdf $g_k(q)$ at iteration k, creates a collection of vectors $\mathbf{q}(k)$. This collection is then used by the measurement process to give an information about the optimum. Via the Kalman estimator, this information is then combined with the pdf $g_k(q)$ in order to produce a new pdf $g_{k+1}(q)$ which will be used in the next iteration. After a sufficient number of iterations, the sequence of estimates (i.e. the $m(k)$) thus produced leads to a near optimal solution.

4.3. Advantages and disadvantages of HKA

HKA shares with some other stochastic algorithms the same interesting features such as: ease of implementation, low memory and CPU speed requirements, search procedure based only on the values of the objective function, no need of strong assumptions such as linearity, differentiability, convexity etc, to solve the optimization problem. In fact it could be used even when the objective function cannot be expressed in an analytic form, in this case, the objective function is evaluated through simulations. However, the main drawback is that HKA may prematurely converge to a local solution, notably when the coefficient α is too high (say about 0.9). The trick is to use low values of this parameter but this lead to a slow convergence of the algorithm. In fact this parameter allows to adjust the trade off between global and local search.

5. Comparison

In this section, the ability of the presented methods to solve a wide range of non-convex optimization problems is tested on various numerical examples, both in the unconstrained and constrained cases. The various experiments were performed using a 1.2 Ghz Celeron personal computer.

5.1. Comparison of HKA with SA, GA and PSO

The optimization methods presented SA, GA, PSO and HKA have been compared using a set of benchmark functions (2 to 30 variables), which are listed in the Appendix A1. In our experiments, the algorithms employed for SA, GA and PSO are adapted from the following MatLab codes: `anneal.m` (Vandekerckhove, 2006), Genetic Algorithm Toolbox (Chipperfield *et al.*, 1995}, and `pso.m` (Devicharan, 2003).

Initialization and parameter setting. The initialization procedure as well as the parameter setting is specific to each optimization method. Table 6 summarizes these issues for SA, GA, PSO and HKA. The parameters used for SA, GA and PSO are essentially those recommended in the literature and presented in sections 1, 2, 3 and 4. Note that one of the great advantages of HKA over other methods is its small number of parameters, only three parameters. Indeed, the parameters of the initial Gaussian distribution are fixed by the bounds of the search space (see Section 4).

Table 6. *Initialization and parameter setting of SA, GA, PSO and HKA.*

Method	Initialization	Parameter setting
SA	The starting point is determined using the method described in section 2.3.1. method 2 i.e. $q_0=\arg\min_{1\le i\le \eta}(J(x^i))$ Initial temperature: $T_0=-(\Delta J)_{max}/\ln(\tau_0)$, where $(\Delta J)_{max}$ is defined as : $(\Delta J)_{max} = \max_{1\le i\le \eta}(J(q^i)) - \min_{1\le i\le \eta}(J(q^i))$ Rule of the decrease of the temperature: $T_{k+1} = 0.8T_k$	Number of samples for the determination of the starting point and the initial temperature: η=4000. Rate of acceptance of $(\Delta J)_{max}$: τ_0=0.8 Final temperature=1e-8. Maximum number of tries within one temperature=100×dim(x) Maximum number of successes within one temperture=12×dim(x)
GA	Initial population is randomly selected in the hyperbox search domain given for each test function (see Appendix).	Population size=100, One point crossover with probability 0.7, probability of mutation=0.07, Precision=16bits, binary coding.
PSO	Initial positions are randomly selected in the hyperbox search domain given for each test function (see Appendix). Initial velocities are randomly selected in the interval $[-m_v, m_v]$.	Swarm size=100, Initial inertia weight=0.8, final inertia weight=0.4, cognitive acceleration factor c_1=2, social acceleration factor c_2=2, Bound for the initial velocities m_v = 4.
HKA	Initial parameters of the gaussian pdf : see section 2.3.1, method 1.	Number of points N=100, Number of best candidates $N_\xi=N/10$, Slowdown coefficient α=0.7

Stopping rules. The algorithm (SA, GA, PSO or HKA) stops when a given number of iterations `MaxIter` is reached (`MaxIter` = 300 in all our experiments[6]) or a given accuracy indicator is obtained. The latest is described for each method hereafter.

[6] To be comparable in term of the maximum number of function evaluations, the maximum number of iterations for SA is `NbPts*MaxIter`, where `NbPts` is the number of points used in our experiments (i.e. the population size (GA), the swarm size (PSO), the value of N (HKA).

SA. The SA stops when the final temperature is reached or after 10 successive temperature stages without any improvement (see section 1).

GA. The GA stops when $\rho_{GA} \leq 0.001$ with $N_G = 10$ (see section 2).

PSO. The PSO stops when $\rho_{PSO} \leq 0.001$ with $N_I = 50$ (see section 3).

HKA. The HKA stops when $\rho_{HKA} \leq 0.005$ (see section 4).

Performance evaluation. To evaluate the methods efficiency, we retained the following criteria summarizing results from 50 minimizations per test function: the success ratio, the mean number of iterations required to obtain a near-optimal solution, the mean computation time and the average error. The success ratio indicates the number of times that the algorithm gives a near-optimal solution for 50 successive runs. In our experiments, the near-optimal set of solutions is defined as $\mathcal{E} = \{q \in \mathcal{D} : J(q) \leq 1.05 J_{min}\}$, where J_{min} is the known optimal solution. The mean number of iterations, the mean computation time and the average error are evaluated in relation to only the successful minimizations (i.e. when a near-optimal solution is found).

The results of SA, GA, PSO and HKA for the test functions F1 to F9 (see Appendix A1) are shown in Table 7. The symbol " - " mean that the algorithm has not converged to a near optimal solution in 50 runs. This occur for SA on test functions F4 (Rastrigin function, 5 variables), F6 (Michalevicz function 10 variables) and F7 (Levy function, 30 variables). The same is true for GA except for the test function F4 with however a very low success ratio. It is clear from Table 7 that the better results are obtained for PSO and HKA.

Table 7. *Comparison of SA, GA, PSO and HKA on test functions F1 to F9.*

	Success Ratio				**Average Number of Iterations (CPU Time second)**				**Average Error to the known global optimum**			
	SA	**GA**	**PSO**	**HKA**	**SA**	**GA**	**PSO**	**HKA**	**SA**	**GA**	**PSO**	**HKA**
F1	39/50	46/50	50/50	50/50	64 (0.4)	66 (1.7)	84 (0.35)	150 (0.5)	2.0e-5	2.5e-5	1.0e-5	1.0e-6
F2	50/50	50/50	50/50	50/50	77 (0.7)	56 (1.7)	128 (0.8)	18 (0.1)	5.0e-3	1.8e-5	1.5e-3	2.0e-4
F3	50/50	50/50	50/50	50/50	81 (2.5)	50 (3.0)	74 (1.5)	46 (0.8)	9.0e-4	7.0e-6	2.0e-6	3.0e-7
F4	–	3/50	42/50	40/50	–	197 (7.5)	174 (0.5)	34 (0.06)	–	3.0e-2	5.0e-8	2.0e-4
F5	44/50	46/50	50/50	49/50	97 (1.3)	135 (7.0)	183 (1.7)	48 (0.4)	1.0e-3	1.0e-2	1.0e-3	1.0e-4
F6	–	–	41/50	43/50	–	–	259 (4.0)	62 (0.9)	–	–	2.5e-1	1.0e-1
F7	–	–	31/50	48/50	–	–	295 (21.0)	83 (5.0)	–	–	6.0e-4	2.0e-3
F8	5/50	49/50	11/50	50/50	87 (1.0)	89 (3.5)	166 (1.0)	76 (0.7)	1.5e-2	7.0e-3	6.5e-2	7.0e-3
F9	16/50	35/50	50/50	48/50	127 (2.5)	56 (2.0)	83 (0.6)	157 (0.9)	5.0e-4	6.0e-4	6.5e-4	7.0e-5

5.2. Comparison of HKA with other metaheuristics

In this section, we complete our numerical experiments by comparing HKA with other metaheuristics. These ones are either enhanced versions of SA, GA and PSO or other methods not discussed in this chapter such as Tabu Search, Ant Colony Optimization or Geometric Programming. We have not programmed the corresponding algorithms but only used the available published results. For the details of these metaheuristics such as the principle of search, the parameter setting, the stopping rule and so forth, we refer to the cited literature. We have considered separately the unconstrained and constrained cases, because specific methods have been proposed to handle constraints (notably the notion of co-evolution or the introduction of an augmented Lagrangian). It must be noted that HKA is the same algorithm in both cases. The constraints are handled merely by introducing an augmented cost function via penalty functions. In all our experiments, the stopping rule and the initialization of the gaussian generator, are the same as those described in the above section.

Unconstrained case. HKA was compared to other metaheuristics such as $ACO_{\mathbf{R}}$, CGA, ECTS, ESA and INTEROPT, which are listed in Table 8. The efficiency of HKA was tested using a set of well known test functions (RC, B2, DJ, $S_{4,5}$, $S_{4,7}$, $S_{4,10}$, and $H_{6,4}$), which are listed in the Appendix A2.

Table 8. *List of the methods used in our comparison.*

Method	Reference
Ant colony optimization for continuous domains ($ACO_{\mathbf{R}}$)	Socha & Dorigo, (2008)
Continuous Genetic Algorithm (CGA)	Chelouah & Siarry, (2000)
Enhanced Continuous Tabu Search (ECTS)	Chelouah & Siarry, (1999)
Enhanced Simulated Annealing (ESA)	P. Siarry *et al.*, (1997)
INTEROPT	Bilbro & Snyder, (1991)

For these experiments we performed each test 100 times and we compared our results with those previously published. In all these experiments, the following parameters have been used: Number of points N = 25, Number of best candidates N_{ξ} = 5, α =0.9. The experimental results are presented in Table 9, for each test function, we give the success ratio for 100 runs and the corresponding average number of function evaluations. It can be seen that some results are not available for ECTS, ESA and INTEROPT (this is indicated by the symbol " - "). From Table 9 we can see that the better results are obtained for $ACO_{\mathbf{R}}$, CGA and HKA. The number of evaluations produced by HKA is slightly larger than those produced by CGA and $ACO_{\mathbf{R}}$, but its ratio of success is better.

Table 9. *Comparison of HKA with $ACO_{\mathbf{R}}$, CGA, ECTS, ESA and INTEROPT.*

	Success Ratio (%)						**Average number of function evaluations**					
	ACO_R	**CGA**	**ECTS**	**ESA**	**INTER OPT**	**HKA**	**ACO_R**	**CGA**	**ECTS**	**ESA**	**INTER OPT**	**HKA**
RC	100	100	100	-	100	100	857	620	245	-	4172	625
B2	100	100	-	-	-	100	559	430	-	-	-	1275
DJ	100	100	-	-	-	100	392	750	-	-	-	600
$S_{4,5}$	57	76	75	54	40	93	793	610	825	1137	3700	675
$S_{4,7}$	79	83	80	54	60	92	748	680	910	1223	2426	686
$S_{4,10}$	81	81	75	50	50	93	715	650	898	1189	3463	687
$H_{6,4}$	100	100	100	100	100	97	722	976	1520	2638	17262	667
Mean Values	89	92	86	65	70	97	684	674	880	1547	6205	745

Constrained case: the welded beam design problem. HKA was compared to other metaheuristics specifically designed for solving constrained problems. In these experiments, HKA handles constraints via a new objective function which includes penalty functions (see relation (2)).

A welded beam is designed for minimum cost subject to constraints on shear stress $\tau(q)$, bending stress in the beam $\sigma(q)$, buckling load on the bar *Pc*, end deflection of the beam $\delta(q)$, and side constraints (Rao, 1996). There are four design variables as shown in Figure 10: *h* (denoted q_1), *l* (denoted q_2), *t* (denoted q_3) and *b* (denoted q_4), $q = [q_1, q_2, q_3, q_4]^T$.

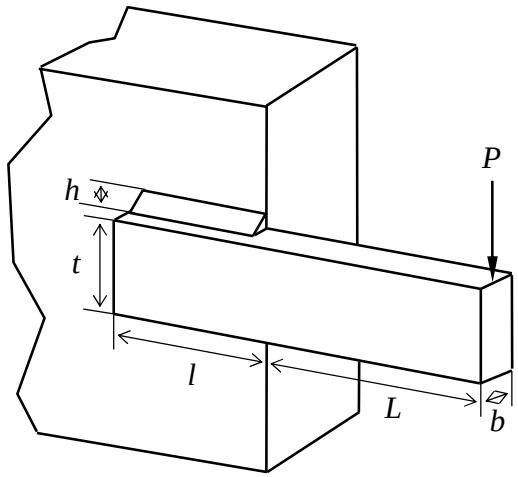

Figure 10 - *Welded beam design problem.*

The problem can be mathematically formulated as follows:

$$
\begin{aligned}
&\text{Minimize} && J(q) = (1+c_1)q_1^2 q_2 + c_2 q_3 q_4 (L+q_2) \\
&\text{Subject to:} && g_1(q) = \tau(q) - \tau_{max} \le 0 \\
& && g_2(q) = \sigma(q) - \sigma_{max} \le 0 \\
& && g_3(q) = q_1 - q_4 \le 0 \\
& && g_4(q) = c_1 q_1 + c_2 q_3 q_4 (L+q_2) - 5 \le 0 \\
& && g_5(q) = h_{min} - q_1 \le 0 \\
& && g_6(q) = \delta(q) - \delta_{max} \le 0 \\
& && g_7(q) = P - Pc(q) \le 0
\end{aligned}
\qquad (29)
$$

Where:

$$
\tau(q) = \sqrt{\tau_1^2 + 2\tau_1\tau_2 \frac{q_2}{2R} + \tau_2^2}, \quad \tau_1 = \frac{P}{\sqrt{2}q_1 q_2}, \quad \tau_2 = \frac{MR}{I}
$$

$$
M = P\left(L + \frac{q_2}{2}\right), \quad R = \sqrt{\frac{q_2^2}{4} + \left(\frac{q_1+q_3}{2}\right)^2}, \quad I = 2\left\{\sqrt{2}q_1 q_2 \left[\frac{q_2^2}{12} + \left(\frac{q_1+q_3}{2}\right)^2\right]\right\} \qquad (30)
$$

$$
\sigma(q) = \frac{6PL}{q_4 q_3^2}, \quad \delta(q) = \frac{4PL^3}{E q_4 q_3^2}, \quad Pc(q) = \frac{4.013E\sqrt{q_3^2 q_4^6/36}}{L^2}\left(1 - \frac{q_3}{2L}\sqrt{\frac{E}{4G}}\right)
$$

and:

$$c_1(q) = 0.10471, \quad c_2(q) = 0.04811, \quad P = 6\times10^3, \quad L = 14, \quad E = 3\times10^7$$
$$G = 1.2\times10^7, \quad h_{min} = 0.125, \quad \delta_{max} = 0.25, \quad \tau_{max} = 1.36\times10^4, \quad \sigma_{max} = 3\times10^4 \tag{31}$$

The ranges of design variables are: $0.1 \le q_1 \le 2, \quad 0.1 \le q_2 \le 10, \quad 0.1 \le q_3 \le 10, \quad 0.1 \le q_4 \le 2$. This problem has been solved by Deb, (1991) using a genetic algorithm (GA) with binary representation, and a traditional penalty function. It has also been solved by Ragsdell & Phillips, (1976) using geometric programming (GP). Recently, this problem has been solved by Coello, (2000) using a GA-based co-evolution model as well as a multi-objective genetic algorithm (MGA) (Coello, 2002; Coello & Montes, 2002). More recently, this problem has also been solved by He and Wang using a co-evolutionary particle swarm optimization (CPSO) and have found a better solution than those previously obtained (He & Wang, 2007).

In this experiment we performed the minimization problem 30 times and we compared our results with those obtained via the methods listed in Table 10. The following parameters have been used: Number of points N = 50, Number of best candidates N_ξ = 5, α = 0.3.

Table 10. *List of the methods used in our comparisons.*

Method	Reference
Geometric Programming (GP)	Ragsdell & Phillips, 1976
Genetic Algorithm and Penalty function (GAP)	Deb, 1991
Co-Evolutionnary Genetic Algorithm (CEGA)	Coello, 2000, 2002
Multi-objective Genetic Algorithm (MGA)	Coello & Montes, 2002
Co-evolutionary Particle Swarm Optimization (CPSO)	He & Wang, 2007

The best solutions obtained by the above-mentioned approaches are listed in Table 11 and the statistical results are shown in Table 12.

Table 11. *Comparison of the best solution found by different methods.*

	GP	**GAP**	**CEGA**	**MGA**	**CPSO**	**HKA**
q_1 (h)	0.245500	0.248900	0.208800	0.205986	0.202369	0.205624
q_2 (l)	6.196000	6.173000	3.420500	3.471328	3.544214	3.473825
q_3 (t)	8.273000	8.178900	8.997500	9.020224	9.048210	9.038561
q_4 (b)	0.245500	0.253300	0.210000	0.206480	0.205723	0.205738
$g_1(q)$	-5743.826517	-5758.603777	-0.337812	-0.074092	-12.839796	-5.621131
$g_2(q)$	-4.715097	-255.576901	-353.902604	-0.266227	-1.247467	-14.103308
$g_3(q)$	0.000000	-0.004400	-0.001200	-0.000495	-0.001498	-0.000115
$g_4(q)$	-3.020289	-2.982866	-3.411865	-3.430043	-3.429347	-3.432290
$g_5(q)$	-0.120500	-0.123900	-0.083800	-0.080986	-0.079381	-0.080624
$g_6(q)$	-0.234208	-0.234160	-0.235649	-0.235514	-0.235536	-0.235550
$g_7(q)$	-3604.275002	-4465.270928	-363.232384	-58.666440	-11.681355	-1.595159
$J(q)$	2.385937	2.433116	1.748309	1.728226	1.728024	1.7255393

Table 12. *Statistical results.*

Method	**Best**	**Mean**	**Worst**	**Std Dev**	**Average number of function evaluations**
GP	2.385937	-	-	-	-
GAP	2.433116	-	-	-	-
CEGA	1.748309	1.771973	1.785835	0.011220	900000
MGA	1.728226	1.792654	1.993408	0.074713	80000
CPSO	1.728024	1.748831	1.782143	0.012926	200000
HKA	1.725539	1.725824	1.726287	0.000172	18600

From Table 11, it can be seen that the best feasible solution found by HKA is better than the best solutions found by other techniques. From Table 12, it can be seen that the average searching quality of HKA is also significantly better than those of other methods, and even the worst solution found by HKA is better than the best solution found via CPSO method. Note also that the standard deviation of the results obtained by HKA is very small. In addition, the number of function evaluations is significantly lower than those obtained by the other methods.

APPLICATION OF STOCHASTIC OPTIMIZATION TO ROBUST STRUCTURED CONTROL AND FAULT DETECTION IN INDUSTRIAL SYSTEMS

1. Robust structured control

The problem of designing a robust controller with a given fixed structure (e.g. a MIMO PID) remains an open issue (Toscano & Lyonnet, 2009). This is mainly due to the fact that the set of all fixed-order/structure stabilizing controllers is non-convex and disconnected in the space of controller parameters. This is a major source of computational intractability and conservatism. Nevertheless, due to their practical importance, some approaches for structured control have been proposed in the literature. Most of them are based on the resolution of Linear Matrix Inequalities LMIs. However, a major drawback with this kind of approaches is the use of Lyapunov variables, whose number grows quadratically with the system size. For instance, if we consider a system of order 70, this requires, at least, the introduction of 2485 unknown variables whereas we are looking for the parameters of a fixed-order/structure controller which contains a comparatively very small number of unknowns. It is then necessary to introduce new techniques capable of dealing with the non-convexity of certain problem arising in automatic control without introducing extra unknown variables. We will show that stochastic methods can be used to this end.

1.1. Formulation of the optimization problem.

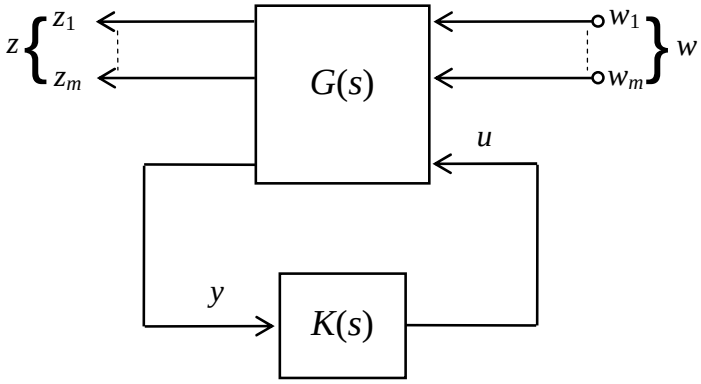

Figure 11 – *Block diagram of the feedback control system.*

Consider the general feedback setup shown in Figure 11, in which $G(s)$ represents the transfer matrix of the process to be controlled

$$\begin{bmatrix} z \\ y \end{bmatrix} = G(s) \begin{bmatrix} w \\ u \end{bmatrix}, \quad \text{with :} \quad G(s) = \left[\begin{array}{c|cc} A & B_1 & B_2 \\ \hline C_1 & D_{11} & D_{12} \\ C_2 & D_{21} & D_{22} \end{array}\right] \tag{32}$$

and $K(s)$ is, for instance, the transfer matrix of a PID controller[7]

$$K(s) = K_p + K_i \frac{1}{s} + K_d \frac{s}{1+\tau s} = \left[\begin{array}{c|c} A_K & B_K \\ \hline C_K & D_K \end{array}\right] = \left[\begin{array}{cc|c} 0 & 0 & K_i \\ 0 & -\frac{1}{\tau} I & -\frac{1}{\tau^2} K_d \\ \hline I & I & K_p + \frac{1}{\tau} K_d \end{array}\right] \tag{33}$$

where K_p is the proportional gain, K_i and K_d are the integral and derivative gains respectively, and τ is the time constant of the filter applied to the derivative action. This low-pass first-order filter ensures the properness of the PID controller and thus its physical realizability. In addition, since $G(s)$ is strictly proper (i.e. it is assumed that $D_{22} = 0$), the properness of the controller ensures the well-posedness of feedback loop.

As depicted Figure 11, the closed-loop system has m external input vectors $w_1 \in \mathbf{R}^{n_{w_1}}, \cdots, w_m \in \mathbf{R}^{n_{w_m}}$ and m output vectors $z_1 \in \mathbf{R}^{n_{z_1}}, \cdots, z_m \in \mathbf{R}^{n_{z_m}}$. Roughly speaking, the global input vector $w = [w_1 \ \dots \ w_m]^T$ captures the effects of the environment on the feedback system; for instance noise, disturbances and references. The global output vector $z = [z_1 \ \dots \ z_m]^T$ contains all characteristics of the closed-loop system that are to be controlled. To this end, the controller $K(s)$ utilizes the measured output vector $y \in \mathbf{R}^{n_y}$, to elaborate the control action vector $u \in \mathbf{R}^{n_u}$ which modify the natural behavior of the process $G(s)$.

The objective is then to determine the PID parameters (K_p, K_i, K_d, τ) allowing to satisfy some performance specifications such as: a good set point tracking, a satisfactory load disturbance rejection, a good robustness to model uncertainties and so one. A powerful way to enforce these kinds of requirements is first to formulate the performance specifications as an optimization problem and then to solve it by an appropriate method. In the $\mathcal{H}_\infty$ framework, the optimization problem can take one of the following forms:

$$\begin{aligned}
&\text{Minimize} && J_\infty(q) = \left\| T_{w_1 z_1}(s,q) \right\|_\infty, \quad q = [\text{vec}(K_p) \ \ \text{vec}(K_i) \ \ \text{vec}(K_d) \ \ \tau]^T \\
&\text{Subject to:} && g_1(q) = \arg\max_{\lambda_i(q)} \{\text{Re}(\lambda_i(q)), \forall i\} - \lambda_{min} \le 0 \\
& && g_2(q) = \left\| T_{w_2 z_2}(s,q) \right\|_\infty - \gamma_2 \le 0 \\
& && \vdots \\
& && g_m(q) = \left\| T_{w_m z_m}(s,q) \right\|_\infty - \gamma_m \le 0
\end{aligned} \tag{34}$$

or also:

$$\begin{aligned}
&\text{Minimize} && J_\lambda(q) = \arg\max_{\lambda_i(q)} \{\text{Re}(\lambda_i(q)), \forall i\}, \quad q = [\text{vec}(K_p) \ \ \text{vec}(K_i) \ \ \text{vec}(K_d) \ \ \tau]^T \\
&\text{Subject to:} && g_1(q) = \left\| T_{w_1 z_1}(s,q) \right\|_\infty - \gamma_1 \le 0 \\
& && g_2(q) = \left\| T_{w_2 z_2}(s,q) \right\|_\infty - \gamma_2 \le 0 \\
& && \vdots \\
& && g_m(q) = \left\| T_{w_m z_m}(s,q) \right\|_\infty - \gamma_m \le 0
\end{aligned} \tag{35}$$

[7] Due to its large diffusion, we consider a PID controller, but the described approach apply for any other fixed structure controller.

where $T_{w_i z_i}(s,q)$ denotes the closed-loop transfer matrix from w_i to z_i, $q \in \mathbf{R}^{n_q}$ is the vector of decision variables regrouping the entries of the matrices K_p, K_i, K_d, and the time constant τ, $\lambda_i(q)$ denotes the i^{th} pole of the closed-loop system and s is the Laplace variable. In the formulation (34) the constraint $g_1(q)$ is required to ensure the stability of the closed-loop system. To do so, the parameter λ_{min} must be set to a negative value.

Note that the formulations (34) and (35) are quite general and can be used to specify many control objectives. For instance, the formulation (34) includes the PID loop-shaping design problem whereas (35) includes the single or mixed sensitivity PID control problem. In the numerical experiments we will see some applications belonging to theses two kind of control problems.

The constrained optimization problem (34) or (35) can be transformed into an unconstrained one, by introducing a new objective function which includes penalty functions (see relation (2) and (3)).

Remark concerning the feasibility issue. In many engineering problems the bounds of a feasible search domain are often known a priori because they are linked to purely material, physical considerations. This is not so clear in control problem for which we have to impose a priori an hyperbox search domain containing stabilizing controllers (i.e. potential solutions of the optimal $\mathcal{H}_\infty$ problem). Finding a priori such a hyperbox is not trivial at all. However, for a given hyperbox search domain it is possible, using HKA, to say whether or not the problem is feasible. More precisely, the feasibility problem can be stated as follows. Given the hyperbox search domain $\mathcal{D} = \{ q_i \in R : \underline{q}_i \le q_i \le \overline{q}_i, \quad i = 1, \cdots, n_q \}$ is there a stabilizing controller? This important issue can be treated via HKA by solving the following optimization problem:

$$\begin{aligned} &\text{Minimize} && J_\lambda(q) = \arg\max_{\lambda_i(q)} \{ \mathrm{Re}(\lambda_i(q)), \forall i \} \\ &\text{Subject to:} && \underline{q}_i \le q_i \le \overline{q}_i, \quad i = 1, \cdots, n_q \end{aligned} \tag{36}$$

where $\lambda_i(q)$ represents the i^{th} pole f the closed-loop system. Let q^* the solution found by HKA to the problem (36). If $J_\lambda(q^*) < 0$, then the problem is feasible within $\mathcal{D}$. In the opposite case, because of the stochastic nature of HKA, this does not necessarily mean that the problem is not feasible (in this case we will say that the problem is probably not feasible).

1.2. Numerical experiments

Mixed sensitivity approach. In this example, for comparison purpose, the same optimization[8] problem as the one presented in Kim, Maruta & Sugie, (2008), is considered:

$$\begin{aligned} &\text{Minimize} && J_\lambda(q) = \arg\max_{\lambda_i(q)} \{ \mathrm{Re}(\lambda_i(q)), \forall i \}, \quad q = [q_1 \;\; q_2 \;\; q_3 \;\; q_4]^T \\ &\text{Subject to:} && g_1(q) = \left\| W_S(s) S(s,q) \right\|_\infty - 1 \le 0 \\ & && g_2(q) = \left\| W_T(s) T(s,q) \right\|_\infty - 1 \le 0 \end{aligned} \tag{37}$$

[8] Note that the optimization problem (37) is of the form (35).

where $q = [q_1 \;\; q_2 \;\; q_3 \;\; q_4]^T$ is the vector of decision variables, s is the Laplace variable, $S(s,q)$ is the sensitivity function defined as $S(s,q) = 1/(1+L(s,q))$, $T(s,q)$ is the closed-loop system defined as $T(s,q) = L(s,q)/(1+L(s,q))$, $L(s,q)$ is the open-loop transfer function defined as $L(s,q) = G(s)K(s,q)$ where $G(s)$ is the transfer function of the system to be controlled and $K(s,q)$ is the transfer function of the PID controller which depends upon the decision variable as follows:

$$K(s,q) = 10^{q_1}\left(1 + \frac{1}{10^{q_2}s} + \frac{10^{q_3}s}{1+10^{(q_3-q_4)}s}\right) \tag{38}$$

Note that the relationship between the decision variables of the optimization problem and the parameters of the PID controller are defined as:

$$K_p = 10^{q_1}, \quad T_i = 10^{q_2}, \quad T_d = 10^{q_3}, N = 10^{q_4} \tag{39}$$

This is done to ensure a broader parameter space of (K_p, T_i, T_d, N). The frequency-dependent weighting functions $W_S(s)$ and $W_T(s)$ are set in order to meet the performance specifications of the closed-loop system. The optimization problem (37), has been solved for the magnetic levitation system described in Sugi, Simizu & Imura, (1993). The process model is defined as:

$$G(s) = \frac{7.147}{(s-22.55)(s+20.9)(s+13.99)} \tag{40}$$

The frequency-dependent weighting functions $W_S(s)$ and $W_T(s)$ are respectively given as:

$$W_S(s) = \frac{5}{s+0.1}, \quad W_T(s) = \frac{43.867(s+0.066)(s+31.4)(s+88)}{(s+10^4)^2} \tag{41}$$

The search space is: $2 \le q_1 \le 4, \quad -1 \le q_2 \le 1, \quad -1 \le q_3 \le 1, \quad 1 \le q_4 \le 3$. In this test, we performed the minimization 30 times and we compared our results with those obtained via ALPSO (Augmented Lagrangian Particle Swarm Optimization see Kim, Maruta & Sugie, 2008). The following parameters have been used: N = 50, N_ξ = 5 and α = 0.4. The best solutions obtained via ALPSO and HKA are listed in Table 13 and the statistical results are shown in Table 14 (the mark "-" means that the corresponding result is not available).

Table 13. *Comparison of the best solutions found via ALPSO and HKA.*

	ALPSO	**HKA**
q_1	3.2548	3.2542
q_2	-0.8424	-0.8634
q_3	-0.7501	-0.7493
q_4	2.3137	2.3139
$g_1(q)$	6.1e-3	-9.6e-4
$g_2(q)$	-4.0e-4	-1.2e-3
$J(q)$	-1.7197	-1.7106

Table 14. *Statistical results.*

Method	Best	Mean	Worst	Std Dev	CPU time	Average number of function evaluations
ALPSO	-1.7197	-	-	-	687 s	25000
HKA	-1.7106	-1.7023	-1.6891	0.0048	266 s	5427

The better value of the objective function obtained with ALPSO is due to the violation of the constraint $g_1(q)$, this is not at all the case in our solution for which all constraints are satisfied. From Table 14 we can observe that the number of function evaluations and the related CPU time are very small compared to ALPSO. It is interesting to note that if, as in ALPSO, a small violation of the constraint $g_1(q)$ is tolerated, we obtain the results listed in Table 15 and Table 16.

Table 15. *Comparison of the best solutions found via ALPSO and HKA.*

	ALPSO	HKA
q_1	3.2548	3.2556
q_2	-0.8424	-0.8354
q_3	-0.7501	-0.7539
q_4	2.3137	2.3127
$g_1(q)$	6.1e-3	4.9e-3
$g_2(q)$	-4.0e-4	-2.8e-3
$J(q)$	-1.7197	-1.7435

Table 16. *Statistical results.*

Method	Best	Mean	Worst	Std Dev	CPU time	Average number of function evaluations
ALPSO	-1.7197	-	-	-	687 s	25000
HKA	-1.7435	-1.7381	-1.7323	0.0030	248 s	5072

From Table 15, it can be seen that the best solution found by HKA is significantly better than the solution found by ALPSO with, in addition, a smaller violation constraint. Table 16, shows that the worst solution found by HKA is better than the solution found via ALPSO, in addition, the number of function evaluations (and so the corresponding CPU time) remains very small compared to ALPSO.

$\mathcal{H}_\infty$ norm minimization. This example is borrowed from Maruta, Kim, & Sugie (2008) which utilises a PSO method for solving the following constrained optimization[9] problem:

$$\begin{aligned} &\text{Minimize} && J_\infty(q) = \left\| T_{wz}(s,q) \right\|_\infty, \quad q = [q_1\ q_2 \cdots q_9]^T \\ &\text{Subject to:} && \max_{\lambda_i(q)} \{ \mathrm{Re}(\lambda_i(q)), \forall i \} < 0 \end{aligned} \tag{42}$$

where $T_{wz}(s,q)$ is the transfer matrix of the closed-loop system, composed by the process $G(s)$ and the controller $K(s)$ (see figure 11). In this example the controller is a first order output feedback described by:

$$K(s,q) = \left[\begin{array}{c|cc} q_1 & q_2 & q_2 \\ \hline q_4 & q_5 & q_6 \\ q_7 & q_8 & q_9 \end{array}\right] \tag{43}$$

[9] Note that the optimization problem (42) is of the form (34).

where q is the decision vector which have to be set in order to satisfy (42). The transfer matrix $G(s)$ of the process to be controlled is given by:

$$G(s)=\left[\begin{array}{ccccccccccccc}
0 & 0 & 1.0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1.5 & -1.5 & 0 & 0.0057 & 1.5 & 0 & 0 & 0 & 0 & 0 & 0 & 0.16 & 0.8 \\
-12.0 & 12.0 & -0.6 & -0.0344 & -12.0 & 0 & 0 & 0 & 0 & 0 & 0 & -19.0 & -3.0 \\
-0.852 & 0.29 & 0 & -0.014 & -0.29 & 0 & 0 & 0 & 0 & 0 & 0 & -0.0115 & -0.0087 \\
0 & 0 & 0 & 0 & -0.73 & 2.8289 & 0 & 0 & 0.1146 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & -1.25 & 0 & 0 & 4.0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & -1000 & 0 & 0 & 1024 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & -1000 & 0 & 0 & 1024 & 0 & 0 \\
\hline
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.01 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.01 \\
\hline
1 & 0 & 0 & 0 & 0 & 0 & -139.0206 & 0 & 0 & 142.8571 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & -139.0206 & 0 & 0 & 142.8571 & 0 & 0
\end{array}\right] \quad (44)$$

The objective is to find a stabilizing q which minimizes the $\mathcal{H}_\infty$ norm of the closed-loop transfer matrix. The search space is: $-15 \le x_i \le 15, \quad i = 1,\ldots 9$. We performed the minimization 35 times (N = 50, N_ξ = 2, α = 0.5) and we compared our results with those obtained in Maruta, Kim, & Sugie (2008) . Tables 17 and 18 summarize de results of this experiment.

Table 17. *Comparison of the best solutions found via PSO and HKA.*

	q_1	q_2	q_3	q_4	q_5	q_6	q_7	q_8	q_9
PSO	-21.1183	-1.5886	11.0822	-2.9907	0.4011	-0.5268	-20.0049	0.4298	-0.9064
HKA	-33.1421	-2.2374	14.8739	-3.2335	0.4748	-0.6776	-25.5573	0.5167	-0.0761

Table 18. *Statistical results (comparison between PSO and HKA).*

Method	Best	Mean	Worst	Std Dev	CPU-time	Nb iter
PSO	1.7092	1.7775	2.3732	0.0996	650 seconds	-
HKA	1.6634	1.7326	2.2377	0.0964	45 seconds	126

From Table 18, we can see that the best solution found by HKA is significantly better than the solution found in Maruta, Kim, & Sugie (2008). The same is true for the mean, worst and standard deviation. In addition, the computation time is very very small compared to PSO.

2. Robust residual generator

In this section, we introduce a simple but effective synthesis strategy for observers based faults detection in linear time-invariant (LTI) systems which are simultaneously affected by two classes of unknown inputs: Noises having fixed spectral densities and unknown finite energy disturbances Khosrowjerdi, Nikoukhah & Safari-Shad, (2005). The problem of designing such an observer, also called a residual generator, will be formulated as a mixed $\mathcal{H}_2/\mathcal{H}_\infty$ optimization problem. This is done to obtain an optimal residual generator, i.e. with minimal sensitivity to unknown inputs. Unfortunately, there is no known solution to this difficult optimization problem. Finding such a residual generator is known to be computationally intractable via the conventional techniques (Toscano & Lyonnet, 2009). This is mainly due to the non-convexity of the resulting optimization problem. To solve this kind of problem easily and directly, without using any complicated mathematical manipulations, we utilize stochastic methods for the resolution of the underlying constrained non-convex optimization problem. A numerical example is given to illustrate the advantage of the mixed $\mathcal{H}_2/\mathcal{H}_\infty$ optimization approach against existing techniques which are based on optimization of $\mathcal{H}_2$ or $\mathcal{H}_\infty$ criteria.

2.1. Formulation of the optimization problem

We assume that the system to be monitored can be described by the following state space model

$$\begin{cases} \dot{x}(t) = Ax(t) + B_u u(t) + B_v v(t) + B_w w(t) + B_f f(t) \\ y(t) = Cx(t) + D_u u(t) + D_v v(t) + D_w w(t) + D_f f(t) \\ x(0) = x_0 \end{cases} \quad (45)$$

where $x \in \mathbf{R}^{n_x}$ is the state vector, $u \in \mathbf{R}^{n_u}$ and $y \in \mathbf{R}^{n_y}$ are, respectively, the known input and output vectors. The unknown input $w \in \mathbf{R}^{n_w}$ represents the process/measurement noises, it is assumed to be of fixed spectral density. The unknown input $v \in \mathbf{R}^{n_v}$ is assumed to be a finite energy disturbance modelling errors caused by exogenous signals, linearization or parameter uncertainties. The unknown input $f \in \mathbf{R}^{n_f}$ is the fault vector; when $f = 0$, system (45) describes the fault-free system (i.e. the normal operating mode). The various constant matrices of (45) are assumed to be known and are of appropriate dimensions. It must be noticed that (45) is an augmented plant model which includes all the weighting functions reflecting the knowledge of w and v. The objective is to develop a residual generator which generates, from the known input/output (i.e. $u(t)$ and $y(t)$), a set of residual signals $r(t)$ that are robust to unknown inputs (i.e. $v(t)$ and $w(t)$) and sensitive to the faults $f(t)$. In these conditions, we can conclude that a fault has occurred if some norm of $r(t)$ is larger than a prespecified threshold or if there are some changes in the statistical properties of the residual signals. This objective can be reached by using an observer-based residual generation. Consider then the following Luenbeger observer-based residual generation:

$$\begin{cases} \dot{z}(t) = Az(t) + B_u u(t) + L(y(t) - D_u u(t) - Cz(t)) \\ r(t) = y(t) - D_u u(t) - Cz(t) \\ z(0) = z_0 \end{cases} \quad (46)$$

where $z \in \mathbf{R}^{n_x}$ is the state vector of the observer and $L \in \mathbf{R}^{n_x \times n_y}$ is the matrix gains to be designed to ensure the stability of the observer as well as the robustness of the residuals to unknown inputs. Combining (45) and (46), we obtain:

$$\begin{cases} \dot{e}(t) = \tilde{A}e(t) + \tilde{B}_v v(t) + \tilde{B}_w w(t) + \tilde{B}_f f(t) \\ r(t) = Ce(t) + D_v v(t) + D_w w(t) + D_f f(t) \\ e(0) = e_0 \end{cases} \quad (47)$$

Where $e = x - z$, $\tilde{A} = A - LC$, $\tilde{B}_v = B_v - LD_v$, $\tilde{B}_w = B_w - LD_w$ and $\tilde{B}_f = B_f - LD_f$. Note that the stability of the residual generator is guaranteed by ensuring that the matrix $\tilde{A}$ is Hurwitz. Taking the Laplace transform of (47), we obtain:

$$r(s) = G_e(s)e_0 + G_v(s)v(s) + G_w(s)w(s) + G_f(s)f(s) \quad (48)$$

where the transfer matrices $G_e(s)$, $G_v(s)$, $G_w(s)$ and $G_f(s)$ are defined as: $G_e(s) = C(sI - \tilde{A})^{-1}$, $G_v(s) = G_e(s)\tilde{B}_v + D_v$, $G_w(s) = G_e(s)\tilde{B}_w + D_w$ and $G_f(s) = G_e(s)\tilde{B}_f + D_f$.

We want the residual r insensitive to unknown inputs and initial conditions, and sensitive to faults. The stability of the observer will ensure the decay to zero of the effect of nonzero initial conditions e_0. Robustness as well as insensitivity to load disturbance can be achieved by satisfying $\|G_v(s,L)\|_\infty \le \gamma$, with γ as small as possible. However, this kind of requirement can also lead to reduction of sensitivity to faults and to an increase of sensitivity to noise. Thus, in addition to $\|G_v(s,L)\|_\infty \le \gamma$, we have to minimize the influence of noise and to maximize the effect of faults. This can be done by solving the following mixed $\mathscr{H}_2/\mathscr{H}_\infty$ optimization problem:

$$\begin{aligned}
&\text{Minimize} \quad J_{mix}(L) = \frac{\|G_w(s,L)\|_2}{\|G_f(s,L)\|_\infty} \\
&\text{Subject to:} \quad g_1(L) = \|G_v(s,L)\|_\infty - \gamma \le 0 \\
&\qquad g_2(L) = \arg\max_{\lambda_i(L)} \{\mathrm{Re}(\lambda_i(L)), \forall i\} - \lambda_{min} \le 0 \\
&\qquad L = [q_{ij}] = \begin{bmatrix} q_{11} & \cdots & q_{1n_y} \\ q_{21} & \cdots & q_{2n_y} \\ \vdots & \vdots & \vdots \\ q_{n_x 1} & \cdots & q_{n_x n_y} \end{bmatrix}, \quad \underline{q} \le q_{i,j} \le \bar{q}
\end{aligned} \tag{49}$$

where $L = [q_{ij}]$ is the matrix of decision variables, $\underline{q}$ and $\bar{q}$ are the bounds of the hyperbox search domain. In the constraint $g_2(L)$, the quantity $\lambda_i(L)$ denotes the i^{th} pole of the observer. The parameter γ, is used to trade off between detection performance and noise sensitivity.

2.2. Numerical experiments

Consider the problem of fault detection in a four-tank process. The state-space process model is given by:

$$\begin{aligned}
\dot{x}(t) = &\begin{bmatrix} -0.0159 & 0 & 0.0419 & 0 \\ 0 & -0.0111 & 0 & 0.0333 \\ 0 & 0 & -0.0419 & 0 \\ 0 & 0 & 0 & -0.0333 \end{bmatrix} x(t) + \begin{bmatrix} 0.0833 & 0 \\ 0 & 0.0718 \\ 0 & 0.0479 \\ 0.0312 & 0 \end{bmatrix} (u(t) + f(t)) \\
&+ \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ -0.0357 & 0 \\ 0 & -0.0313 \end{bmatrix} v(t), \quad y(t) = \begin{bmatrix} 0.5 & 0 & 0 & 0 \\ 0 & 0.5 & 0 & 0 \end{bmatrix} x(t) + w(t)
\end{aligned} \tag{50}$$

where the state vector $x = [x_1\ x_2\ x_3\ x_4]^T$ represents the level of water in the tanks, the control input $u = [u_1\ u_2]^T$ is the voltage applied to the pumps, $f = [f_1\ f_2]^T$ is the fault vector associated to the pumps, $v = [v_1\ v_2]^T$ is the disturbance vector and $w = [w_1\ w_2]^T$ is the measurement noise vector. The objective is to detect the actuator fault f in the presence of a disturbance v and the measurement noise w. To evaluate the performance of the residual generator, a fault f_1 and a disturbance v_1 are applied (see Figure 12).

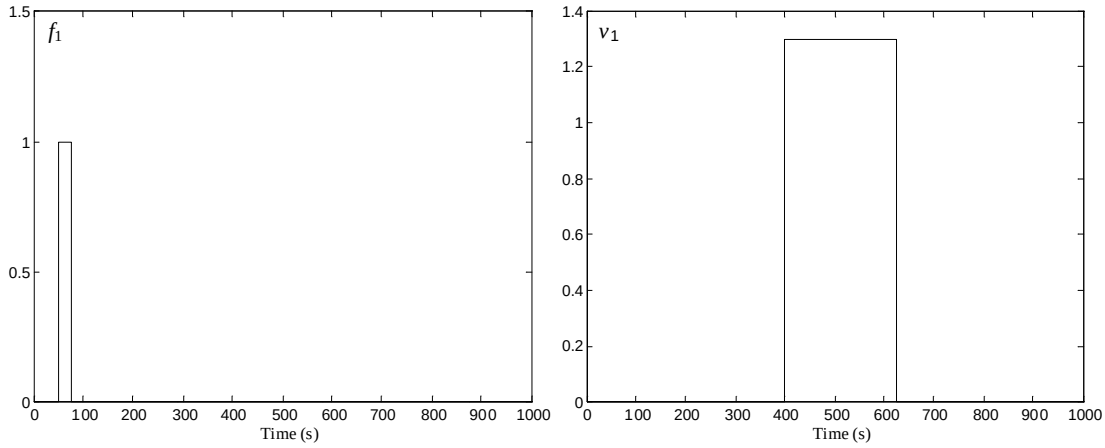

Figure 12 – *Pump fault and disturbance*.

The synthesis of the mixed $\mathscr{H}_2/\mathscr{H}_\infty$ residual generator was done by solving the optimisation problem (49) via HKA. The following parameters have been used: $N = 50$, $N_\xi = 5$, $\alpha = 0.4$, $\underline{q} = -1$, $\overline{q} = 1$, $\lambda_{min} = -0.01$ and $\gamma = 0.08$. Figure 13 shows the simulation result obtained with the resulting residual generator. This figure describes the evolution of the absolute value of the residual $r_1(t)$. We can see that the effect of the disturbance $v_1(t)$ on the residual $r_1(t)$ is strongly attenuated and the effect of the fault is significantly bigger than that of $v_1(t)$. Therefore, this fault can be easily detected by using an appropriate threshold. The ratio between the maximum values of the effect of the fault to the maximum value of the effect of the disturbance is 2.6. This ratio is only of 1.8 by using the approach proposed by Khosrowjerdi *et al.* (2005). This clearly shows the better performance of the proposed approach. Indeed, HKA allows to solve directly the optimisation problem (49) without using any upper bound nor transforming the non-convex problem into a convex one, as it is the case in Khosrowjerdi *et al.* (2005).

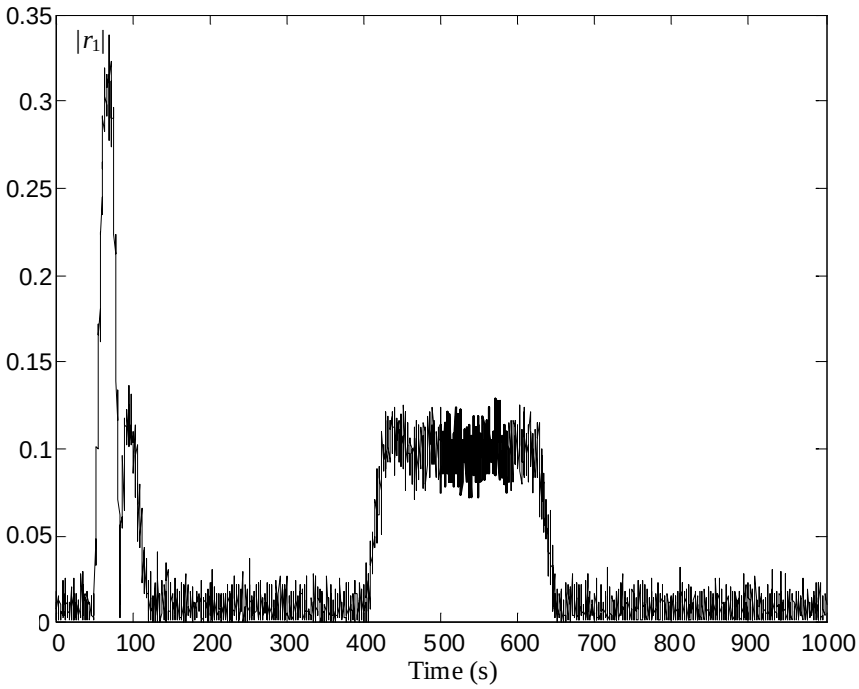

Figure 13 – *Evolution of* $|r_1(t)|$, *mixed* $\mathscr{H}_2/\mathscr{H}_\infty$, $\gamma = 0.08$.

For comparison, Figure 14 shows the result obtained when the residual generator is designed by just solving the $\mathcal{H}_2$ optimisation problem:

$$\begin{cases} L_{opt} = \arg\min_{L} \left\| \begin{bmatrix} G_v(s,L) & G_w(s,L) \end{bmatrix} \right\|_2 \\ g(L) = \arg\max_{\lambda_i(L)} \{ \mathrm{Re}(\lambda_i(L)), \forall i \} - \lambda_{min} \le 0 \end{cases} \tag{51}$$

Similarlely, Figure 14 shows the result obtained by just solving the $\mathcal{H}_\infty$ optimisation problem

$$\begin{cases} L_{opt} = \arg\min_{L} \left\| \begin{bmatrix} G_v(s,L) & G_w(s,L) \end{bmatrix} \right\|_\infty \\ g(L) = \arg\max_{\lambda_i(L)} \{ \mathrm{Re}(\lambda_i(L)), \forall i \} - \lambda_{min} \le 0 \end{cases} \tag{52}$$

As we can see, the corresponding residual generators cannot be used to detect the fault $f_1(t)$. This confirms the usefulness of a mixed $\mathcal{H}_2/\mathcal{H}_\infty$ synthesis.

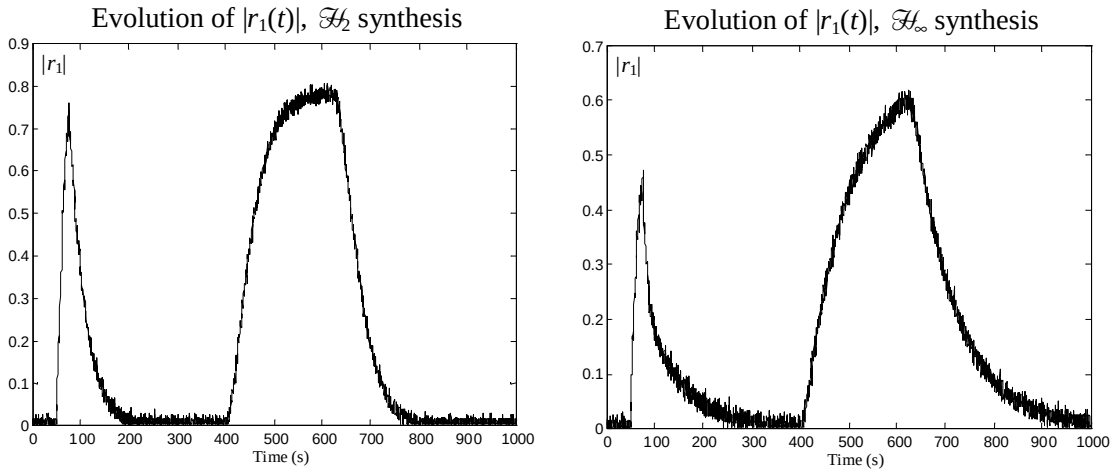

Figure 14 – *Evolution of* $|r_1(t)|$*, in a* $\mathcal{H}_2$ *and* $\mathcal{H}_\infty$ *framework respectively.*

CONCLUSION

It is a matter of fact that Nature has been, and is always, a major source of inspiration for scientific and technical developments. Optimization does not escape to this rule and many heuristic searches draw their foundations from physical or biological principles such as the main approaches reviewed in this chapter. Although they are pale imitations of the reality, these approaches have proven their efficiency in solving difficult optimizations problems. One of the main purposes with this chapter was to provide the essential ideas behind each presented optimization method as well as the algorithm and the usually adopted parameter setting. This could help the reader in the practical use of these methods. In addition to the standard stochastic algorithm, we have presented the recently developed optimization method called HKA. This new approach has been compared favourably with many other metaheuristics on several test problems, both in the unconstrained and constrained cases. No general conclusion can be drawn from these few numerical experiments, but it seems that, in some cases, HKA is a good alternative for solving difficult non-convex problems.

The efficiency of stochastic methods in solving difficult non-convex problem has been shown on many practical examples. Notably, we have addressed the problems of robust structured control and fault diagnosis of industrial systems. These topics lead indeed to non-convex constrained optimization problems which are known to be difficult to deal with using conventional methods. Since stochastic methods does not require strong assumptions such as linearity, differentiability, convexity etc. they can be used to find out, in a straightforward manner, if not the optimal solution but at least a suboptimal one, which is very useful for the practitioner.

APPENDIX

A1. List of test functions F1 to F9

Easom's function (F1) (2 variables): $J(q) = -\cos(q_1)\cos(q_2)\exp(-((q_1-\pi)^2 + (q_2-\pi)^2))$
search domain: $-100 \le q_i \le 100$, $i = 1, 2$; global minimum: $q_{op\,t} = (\pi, \pi)$, $J(q_{op}) = -1$.

Goldstein-Price's function (F2) (2 variables); search domain: $-2 \le q_i \le 2$, $i = 1, 2$; global minimum: $q_{opt} = (-1, 0)$, $J(q_{op}) = 3$. For a complete definition of this function see Socha & Dorigo, (2008).

Hartmann's function (F3) (3 variables); search domain: $0 \le q_i \le 1$, $i = 1, 3$; global minimum: $q_{opt} = (0.1146, 0.5556, 0.8525)$, $J(q_{op}) = -3.86278$. For a complete definition of this function see Socha & Dorigo, (2008).

Rastrigin's function (F4) (5 variables)
$J(q) = 50 + \sum_{i=1}^{5}(q_i^2 - 10\cos(2\pi q_i))$
search domain: $-3 \le q_i \le 3$, $i = 1, 5$; global minimum: $q_{op\,t} = 0$, $J(q_{op}) = 0$.

Zakharov's function (F5) (5 variables)
$J(q) = \sum_{i=1}^{5} q_i^2 + \left(\sum_{i=1}^{5} 0.5 i q_i\right)^2 + \left(\sum_{i=1}^{5} 0.5 i q_i\right)^4$
search domain: $-6 \le q_i \le 12$, $i = 1, 5$; global minimum: $q_{op\,t} = 0$, $J(q_{op}) = 0$.

Michalevicz's function (F6) (10 variables)
$J(q) = -\sum_{i=1}^{10}\left(\sum_{i=1}^{5}\sin(q_i)\sin\left(\frac{i q_i^2}{\pi}\right)\right)^{20}$
search domain: $0 \le q_i \le \pi$, $i = 1, 10$; global minimum: $J(q_{op}) = 9.66015$.

Levy's function (F7) (30 variables)

$$z_i = 1 + \frac{q_i - 1}{4}, \quad i = 1, \cdots, 30$$

$$J(q) = \sin^2(\pi z_1) + \sum_{i=1}^{29}[(z_i - 1)^2(1 + 10\sin^2(\pi z_i + 1))] + (z_{30} - 1)^2(1 + \sin^2(2\pi z_{30} + 1))$$

search domain: $-10 \le q_i \le 10$, $i = 1, 30$; global minimum: $q_{op\,t} = (1,1,\ldots,1)$, $J(q_{op}) = 0$.

Constrained test problem (F8) (3 variables)

Maximise $J(q) = \dfrac{3q_1 + q_2 - 2q_3 + 0.8}{2q_1 - q_2 + q_3} + \dfrac{4q_1 - 2q_2 + q_3}{7q_1 + 3q_2 - q_3}$

Subject to:
$g_1(q) = q_1 + q_2 - q_3 - 1 \le 0$
$g_2(q) = 12q_1 + 5q_2 + 12q_3 - 34.8 \le 0$
$g_3(q) = -6q_1 + q_2 + q_3 + 4.1 \le 0$
$g_4(q) = -q_1 + q_2 - q_3 + 1 \le 0$
$g_5(q) = 12q_1 + 12q_2 + 7q_3 - 29.1 \le 0$

search domain: $0 \le q_i \le 10$, $i = 1, 3$; global minimum: $q_{op\,t}$= (1,0,0), $J(q_{op})$ = 2.471428$.

Constrained test problem (F9) (5 variables)

Minimise $J(q) = \exp(q_1 q_2 q_3 q_4 q_5)$

Subject to:
$g_1(q) = q_1^2 + q_2^2 + q_3^2 + q_4^2 + q_5^2 - 10 \le 0$
$g_2(q) = q_1 q_3 - 5q_4 q_5 = 0$
$g_3(q) = 1 + q_1^3 + q_2^3 = 0$

search domain: $-2.3 \le q_i \le 2.3$, $i = 1, 2$; $-3.2 \le q_i \le 3.2$, $i = 3, 4, 5$; global minimum: $q_{op\,t}$= (-1.717143, 1.595709, 1.827247, -0.7636413, -0.7636450), $J(q_{op\,t})$ = 0.053950.

A.2. Test functions RC, B2, DJ, $S_{4,5}$, $S_{4,7}$, $S_{4,10}$, and $H_{6,4}$

Branin's function (RC) (2 variables)

$$J(q) = \left(q_2 - \frac{5.1}{4\pi^2} q_1^2 + \frac{5}{\pi} q_1 - 6 \right)^2 + 10\left(1 - \frac{1}{8\pi}\right)\cos q_1 + 10$$

search domain: $-5 \le q_i \le 10$, $i = 1, 2$; three global minima $q_{op\,t}$= ($-\pi$, 12.275), (π, 2.275), (9.42478, 2.475), $J(q_{op\,t})$ = 0.397887.

Bohachecsky's function (B2) (2 variables)

$J(q) = q_1^2 + 2q_2^2 - 0.3\cos(3\pi q_1) - 0.4\cos(4\pi q_2) + 0.7$

search domain: $-100 \le q_i \le 100$, $i = 1, 2$; global minimum $q_{op\,t}$= (0, 0), $J(q_{op\,t})$ = 0.

De Jong's function (DJ) (3 variables)

$J(q) = q_1^2 + q_2^2 + q_3^2$

search domain: $-5 \le q_i \le 5$, $i = 1, 3$; global minimum $q_{op\,t}$= (0, 0, 0), $J(q_{op\,t})$ = 0.

Shekel's functions $S_{4,5}$, $S_{4,7}$, $S_{4,10}$ (4 variables); search domain: $0 \le q_i \le 9$, $i = 1, 4$; 3 functions were considered $S_{4,5}$, $S_{4,7}$, $S_{4,10}$. For a complete definition of these function see Socha & Dorigo, (2008).

Hartmann's functions $H_{6,4}$ (6 variables); search domain: $0 \le q_i \le 1$, $i = 1, 6$; global minimum: $J(q_{op\,t})$ = -3.322368. For a complete definition of this function see Socha & Dorigo, (2008).

REFERENCES

Angeline, P. J. (1998). Using selection improve particle swarm optimization. In Proceedings of the *IEEE International Conference on Evolutionary Computation* (pp. 84-89). Piscataway, New Jersey, USA: IEEE Press.

Ben-Ameur, W. (2004). Computing the Initial Temperature of Simulated Annealing. *Computational Optimization and Applications*, 29(3), 369-385.

Bilbro, G. L., & Snyder, W.E. (1991). Optimization of Functions with Many Minima. *IEEE Transactions on Systems, Man, and Cybernetics*, 24(4), 840-849

Blum, C., & Li, X. (2008). Swarm Intelligence in Optimization. In C. Blum & D. Merkle (Eds.), *Swarm Intelligence Introduction and Applications* (pp. 43-85). Springer Berlin Heidelberg.

Bohachevsky, I. O., Johnson, M. E., & Stein M. L. (1986). Generalized simulated annealing for function optimization. *Technometrics*, 28(3), 209-217.

Boyd, S., & Vandenberghe, L. (2004). *Convex optimization*. New York: Cambridge University Press.

Boyd, R., & Richardson, P. (1985). *Culture and the evolutionary process*. Chicago: University of Chicago Press.

Brook, S. P., & Morgan, B. J. T. (1995). Optimization using simulated annealing. *The statistician*, 44(2), 241-257.

Cerny, V. (1985). A thermodynamical approach to the travelling salesman problem: an efficient simulation algorithm. *Journal of Optimization Theory and Applications*, 43(1), 41-51.

Chelouah, R., & Siarry, P. (1999). Enhanced Continuous Tabu Search: An Algorithm for the Global Optimization of Multiminima Function. In S. Voss, S. Martello, I.H. Osman & C. Roucairol (Ed.), *Meta-Heuristics, Advances and Trends in Local Search Paradigms for Optimization* (pp. 49-61). Kluwer Academic Publishers.

Chelouah, R., & Siarry, P. (2000). A continuous genetic algorithm designed for the global optimization of multimodal functions. *Journal of Heuristics*, 6(2), 191-213.
year=2000}

Chipperfield, A., Fleming, P., Pohlheim, H., & Fonseca, C. (1995). Genetic Algorithm TOOLBOX For Use with MATLAB.

Clerc M., & Kennedy, J. (2002). The particle swarm-explosion, stability, and convergence in a multidimensional complex space. *IEEE Transactions on Evolutionary computation*, 6(1), 58-73.

Coello, C.A.C. (2000). Use of a self-adaptive penalty approach for engineering optimization problems. *Computers in Industry*, 41(2), 113-127.

Coello, C.A.C. (2002). Theoretical and numerical constraint handling techniques used with evolutionary algorithms: a survey of the state of the art. *Computer Methods in Applied Mechanics and Engineering*, 191(11-12), 1245-1287.

Coello, C.A.C. & Montes, E.M. (2002). Constraint-handling in genetic algorithms through the use of dominance-based tournament selection. *Advanced Engineering Informatics*, 16(3), 193-203.

Corana, A., Marchesi, M., Martini, C., & Ridella, S. (1987). Minimising Multimodal Functions of Continuous Variables with the Simulated Annealing Algorithm. *ACM Transactions on Mathematical Software*, 13(3), 262-280.

Deb, K. (1991). Optimal design of a welded beam via genetic algorithms. *AIAA Journal*, 29(11), 2013-2015.

Devicharan, D., (2003). pso.m. Available at http://web.syr.edu/~ddevicha/pso.m.

Dréo, J., Pétrowski, A., Siarry, P., & Taillard, E. (2006). *Metaheuristic for hard optimization*. Berlin: Springer-Verlag.

Eiben, A.E., & Schoener, M. (2002). Evolutionary computing. *Information processing letter*, 82(1), 1-6.

Fogel, D.B. (2006). *Evolutionary computation: towards a new philosophy of machine intelligence.* Wiley-IEEE Press.

Garey, M. R., & Johnson, D. S. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. San Francisco: Freeman, CA.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*. Kluwer Academic Publishers, Boston, MA.

Hajek, B. (1988). Cooling schedules for optimal annealing. *Mathematics of Operation Research*, 13(2), 311-329.

He, Q., & Wang, L. (2007). An effective co-evolutionary particle swarm optimization for constrained engineering design problems. *Engineering Applications of Artificial Intelligence*, 20(1), 89-99.

Holland, J. H. (1962). Outline for logical theory of adaptive systems. *Journal of the ACM*, 9(3), 297-394.

Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems*. Ann Arbor: University of Michigan Press.

Ingber, L. (1994). Simulated annealing: practice versus theory. *Math. Comput. Modelling*, 18(11), 29-57.

Kennedy, J., & Eberhart, R. C. (1995). Particle swarm optimization. In Proceedings of *IEEE International Conference on Neural Networks* (pp. 1942-1948). Piscataway, NJ, USA: IEEE Press.

Kim, T.H., Maruta, I., & Sugie, T. (2008). Robust PID controller tuning based on the constrained particle swarm optimization. *Automatica*, 44(4), 1104-1110.

Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. P. (1983). Optimization by Simulated Annealing. *Science*, 220(4598), 671-680.

Khosrowjerdi, M. J., Nikoukhah, R., & Safari-Shad N. (2005). Fault detection in a mixed H2/Hinf setting. *IEEE Transactions on Automatic Control*, 50(7), 1063-1068.

Lobo, F., Lima, C. F., & Michalewicz, Z. (Eds.). (2007). *Parameter Setting in Evolutionary Algorithms. Studies in Computational Intelligence.* Berlin, Springer Verlag.

Locatelli, M. (2000). Convergence of a simulated annealing algorithm for continuous global optimization. *Journal of Global Optimization*, 18(3), 219-234.

Lovberg, M., & Krink, T. (2002). Extending particle swarm optimizers with self-organized criticality. In Proceedings of the *fourth congress on evolutionary computation, Vol. 2* (pp. 1588-1593).

Maruta, L., Kim, T.H., & Sugie, T. (2008). Synthesis of fixed-structure Hinf, controllers via Constrained Particle Swarm Optimization. *Proc. of the 17th IFAC World Congress*, Seoul, Korea, pp. 7843-7848.

Maybeck, P. S. (1979). *Stochastic models, estimation, and control*. New-York: Academic Press.

Miller, B. L., & Goldberg, D. E. (1995). Genetic algorithms, tournament selection, and the effects of noise. *Complex Systems*, 9(3), 193-212.

Metropolis, N. , Rosenbluth, A.W., Rosenbluth, M.N., Teller, A.H., & Teller, E. (1953). Equations of state Calculations by Fast Computing Machines. *Journal of Chemical Physics*, 21(6):1087-1092.

Michalewicz, Z. (1998). *Genetic Algorithms + Data Structures = Evolution Programs*. Springer-Verlag, Berlin.

Parks, G.T. (1990). An intelligent stochastic optimization routine for nuclear fuel cycle design. *Nucl. Technol.*, 89(2), 233-246.

Pincus, M. 1970. A Monte Carlo Method for the Approximate Solution of Certain Types of Constrained Optimization Problems. *Oper. Res*, 18, 1225-1228.

Ragsdell, K.M., & Phillips, D.T. (1976). Optimal design of a class of welded structures using geometric programming. *ASME Journal of Engineering for Industries*, 98(3), 1021-1025.

Rao, S. S. (1996). *Engineering Optimization*. New York: Wiley.

Rockafellar, R. T. (1993). Lagrange multipliers and optimality. *SIAM Review*, 35(2), 183-238.

Siarry, P., Berthiau, G., Durbin, F., & Haussy, J. (1997). Enhanced Simulated Annealing for Globally Minimizing Functions of Many Continuous Variables. *ACM Transactions on Mathematical Software*, 23(2), 209-228.

Socha, K., & Dorigo, M. (2008). Ant colony optimization for continuous domains. *European Journal of Operational Research*, 185(3), 1155-1173.

Spall, J. C. (2003). *Introduction to stochastic search and optimization*. Wiley-Interscience, John Wiley & Sons.

Sugi, T., Simizu, K., & Imura, J. (1993). Hinf control with exact linearization and its applications to magnetic levitation systems. In *IFAC 12th World congress*, Vol. 4, 363-366.

Toscano, R., & Lyonnet, P. (in press). Heuristic Kalman Algorithm for solving optimization problems. *IEEE Transaction on Systems, Man, and Cybernetics, Part B.*

Toscano, R., & Lyonnet, P. (in press). Mixed H2/Hinf residual generator design via Heuristic Kalman Algorithm. Safeprocess'09, Barcelone, Spain.

Toscano, R., & Lyonnet, P. (in press). Robust PID controller tuning based on the Heuristic Kalman Algorithm. *Automatica*.

Vandekerckhove, J. (2006). anneal.m. Available at http://pages.stern.nyu.edu/~acollard/anneal.m.

Vanderbilt, D., & Louie, S. G. (1984). A Monte Carlo simulated annealing approach to optimization over continuous variables. *Journal of Computational Physics*, 56(2), 259-271.