UNIVERSITÀ DEGLI STUDI DI PARMA

Dottorato di Ricerca in Tecnologie dell'Informazione

XXVII Ciclo

META-OPTIMIZATION OF BIO-INSPIRED TECHNIQUES FOR OBJECT RECOGNITION

Coordinatore:

Chiar.mo Prof. Marco Locatelli

Tutor:

Chiar.mo Prof. Stefano Cagnoni

Dottorando: *Roberto Ugolotti*

Gennaio 2015

Alla mia famiglia:
a chi c'è da sempre,
a chi non c'è più
e a chi è arrivato da poco.

Summary

List of Figures

List of Tables

List of Algorithms

Introduction

Object recognition is the task of automatically finding a given object in an image or in a video sequence. This task is very important in many fields such as medical diagnosis, advanced driving assistance, image understanding, surveillance, virtual reality. Nevertheless, this task can be very challenging because of artefacts (related with the acquisition system, the environment or other optical effects like perspective, illumination changes, etc.) which may affect the aspect even of easy-to-identify and well-defined objects.

A possible way to achieve object recognition is using model-based approaches: in this scenario a model (also called template) representing the properties of the target object is created; then, hypotheses on the position of the object are generated, and the model is transformed accordingly, until the best match with the actual appearance of the object is found.

To generate these hypotheses intelligently, a good optimization algorithm is required. Bio-inspired techniques are optimization methods whose foundations rely on properties observed in nature (such as cooperation, evolution, emergence). Their effectiveness has been proved in many optimization tasks, especially in multi-modal, multi-dimensional hard problems like object recognition.

Although these heuristics are generally effective, they depend on many parameters that strongly affect their performances; therefore, a significant effort must be spent to understand how to let them express their full potentialities.

This thesis describes a method to (i) automatically find good parameters for bio-inspired techniques, both for a specific problem and for more than one at the same time, and (ii) acquire more knowledge of a parameter's role in such algorithms. Then, it shows how bio-inspired techniques can be successfully applied to different object recognition tasks, and how it is possible to further improve their performances by means of automatic parameter tuning.

The remainder of the thesis is organized as follows:

- Chapter 1 provides the necessary theoretical background to follow the remainder of the paper. It is not meant to be a comprehensive review but to allow the reader to focus on the main topics of the thesis (bio-inspired optimization and object recognition); nevertheless many references are provided for further documentation on each single topic;
- Chapter 2 describes the main motivations behind automatic parameter tuning and summarizes the main approaches used to tackle this problem;
- Chapters 3 and 4 describe the two versions (single and multi-objective) of the meta-optimization method developed, as well as several tests and experiments that prove their effectiveness;
- Chapter 5 shows some applications in which model-based object recognition has been employed, and how the automatic parameter tuning techniques have been employed to further improve their performances;
- Lastly, Chapter 6 summarizes the contents of this thesis;
- This work is followed by an Appendix that describes the GPGPU computing paradigm and a library developed to exploit it. It is not necessary in order to understand the rest of the work, but it may be of help in reproducing the experiments described in this thesis.

Chapter 1

Theoretical Background

Avoid both "practically hopeless, although entirely correct"
and "it always works like magic, no need for theory" approaches.

– János D. Pintér

This Chapter introduces the two main topics of this dissertation, Global Continuous Optimization and Object Recognition. A broad description of these topics will be provided, followed by a more in depth description of the algorithms and strategies employed in the rest of the work.

1.1 Global Continuous Optimization

Every day, in many different engineering and industry fields, several decisions need to be taken. Let us consider some examples:

- A novel axial fan needs to be designed, the main characteristic requested being its efficiency. The designer has a very precise modeling software that allows to predict the efficiency according to to how the fan's parameters have been set. The fan's parameters on which the designer can directly operate contain blades' number, thickness and angle, hub's size, and others. Therefore, the job the designer is requested to do is to find a feasible combination of parameters that maximize the efficiency;

- Color quantization of an image consists of creating a palette with a limited number of colors onto which each color in an image can be mapped. In this case, the goal is to find the set of colors that reduce the overall difference between the original and the quantized images;

- A medical doctor has a large database containing the outcome of several exams performed on each patient. Some of these exams give indirect hints about the health of each patient, while other exams (more complex, intrusive, or costly) tell if the patient has or does not have a certain disease. The doctor would like to know if the results of the faster and harmless exams could be used to predict the results of the slower and more intrusive ones; in other words he wants to find the set of parameters that are more likely to predict the health status of the patient as fast as possible, at the lowest cost and with the smallest possible side effects.

These situations, although very different from one another, have a weak element in common: they all have some information that comes in input and they have to produce a decision in output accordingly.

Consequently, finding the solution of any problem like these can be modeled as finding the global best of a function, which may represent the quality, error, cost, utility, etc. of a possible solution. In some lucky cases, this function may have a simple behavior and the global best can be found using local techniques or fast “trial and error” approaches. But, in general, a function modeling a real world phenomenon has a non-linear, non-convex behavior, which requires techniques able to widely explore the function domain to be dealt with.

Global Continuous Optimization (GCO) [59] consists of finding the overall best solution for a problem under consideration.

Some notations

A GCO problem can be described using simple mathematical notation (from now on a minimization problem will be considered, but any such problem can be turned into a maximization one, without loss of generality, by simply changing the sign of the function):

$$\vec{x_{min}} = \underset{\vec{x} \in D}{\arg\min}\, f(\vec{x})$$

where $\vec{x} \in \mathbb{R}^n$ is a possible decision vector, solution or input, n being the number of input components, also called dimensionality of the problem, and $\vec{x_{min}}$ is a solution vector (there can be a single one or many of them) for which the function f reaches its minimum within D. $D \in \mathbb{R}^n$ is the domain (also called search space) of the function, which means the set of all possible solutions. D may be limited by lower and upper limits that may correspond to physical constraints or valid ranges of the variable under consideration ($l_i \leq x_i \leq u_i, i = 1, \ldots, n$), as well as by other constraints, which may represent combinations of input variables that violate some constraints (mathematical, physical or of other nature) imposed on the solution.

The function $f : \mathbb{R}^n \rightarrow \mathbb{R}$ is the objective function and usually does not need to have any particular property, such as convexity, continuity or differentiability. In some scenarios, the function does not even have to be known: this is the case of “black-box” optimization, where the function is used as a “black box” in which the only information known is the output that corresponds to a given input. Anyway, the properties of f have a great impact on the tractability of the problem.

A local minimum $\vec{m}$ of a function is a point for which:

$$f(\vec{m}) \leq f(\vec{x}) \quad \forall \vec{x} \in N_{\vec{m}}$$

where $N_{\vec{m}} = \{\vec{x} \in D : |\vec{x} - \vec{m}| < \varepsilon\}, \varepsilon > 0$, is a neighborhood of $\vec{m}$. If the function has only one local minimum (which, in this case, is also the global minimum), it is called *unimodal*, in the opposite case it is *multimodal*. It is evident that in the latter case the search for the best can be misled by the local minima (see Figure 1.1), while, in the first case, it is sufficient to "follow" the slope of the function to reach $\vec{x_{min}}$.

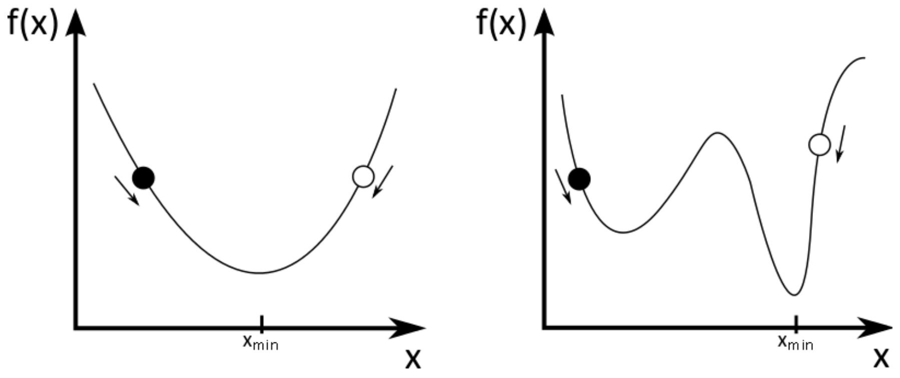


Figure 1.1: The two graphs show an example of a unimodal function (on the left) and of a multimodal one (on the right). On the x axis, the input vector (in this case it is one-dimensional), on the y-axis the value of the objective function. In the first case, starting from both points and following the slope of the function will lead to the global optimum x_{min}. In the second one, only the white point will lead eventually to the global optimum while, if the optimizer starts from the black point, it will find a sub-optimal solution.

Another important property of a function in a GCO problem is *separability*. A function is said to be (additively) separable if it can be decomposed as follows:

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

If a function respects this equality, each element of the solution vector can be analyzed independently of the others, which simplifies the task; otherwise (*non-separable* functions) the search strategy needs to take into consideration also the dependencies between the elements that compose a solution.

GCO Strategies

A GCO strategy usually involves an iterative process within which the solution is refined over time. The strategies used to solve GCO problems can be divided in three main groups (see also Figure 1.2):

- Deterministic approaches [42]: they are based on the mathematical properties of the function (e.g. gradient, Hessian matrix) and guarantee the discovery of at least one global optimum. The main drawback is their computational complexity (usually exponential against problem dimensionality) which can make them unusable in many problem classes, such as NP-hard ones;
- Exact stochastic methods [197]: these methods (as well as Soft Computing ones, see next point) do not guarantee that the global best solution is found, but good methods are able to reach reasonably good approximations of $\vec{x_{min}}$;
- Soft Computing methods: these methods often offer solutions with a lower computational burden than the deterministic ones. They are more suited for NP-hard problems for which no deterministic algorithm can compute a solution in a feasible time. The remainder of this Chapter will focus on these methodologies.

1.1.1 Soft Computing

The term Soft Computing (SC) has been introduced in the early eighties and comprehends a vast number of techniques which can perform several tasks like classification, data mining, and optimization. According to Zadeh [192], the main ability of SC is to “mimic the human mind to effectively employ modes of reasoning that are approximate rather than exact”. SC techniques aim at formalizing the humans’ ability to tackle problems that contain uncertainty, errors, or missing information by learning from examples, previous knowledge or experience and cooperation.

SC techniques have been applied in many tasks, such as classification [53, 130], prediction [4], clustering [60], data mining [108]. When considering GCO problems, the most successful techniques fall under the name of metaheuristics.

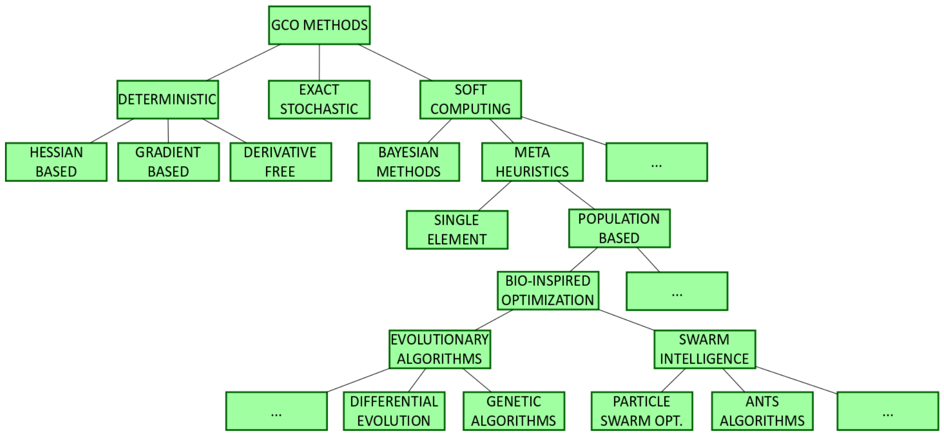


Figure 1.2: A partial taxonomy of GCO methods, focused on the methods studied in this thesis.

1.1.2 Metaheuristics

Fred Glover introduced the term metaheuristic (MH) in [48]. Many definitions have been proposed to identify this term since then. One of the most concise and precise, although flexible enough to adapt to several techniques, has been proposed by Osman and Laporte [128]. They define a MH as an "iterative generation process which guides a subordinate heuristic by combining intelligently different concepts [...] in order to find efficiently near-optimal solutions". To be able to do so, they intrinsically need to have some abilities. One of the most useful is to efficiently explore the search space without getting trapped in a limited area while, on the opposite, being able to deeply analyze the most promising parts of the search space. Therefore, they must be able to cope with two different phases of a search process:

- exploration (or diversification) is the part of a search process in which new areas of the search space are considered, in order to find new promising zones;
- exploitation (or intensification), on the contrary, consists of focusing on a limited area with the goal to improve a current good but non-optimal solution.

The balance of these two phases is one of the most important keys to the success of an algorithm: too much emphasis on the first one makes the MH wander across the search space (an extreme scenario is a “random walk”); on the opposite side, if exploitation is overly privileged, it may miss good areas of the function domain and land on local minima (extreme case: local search). Each MH has a different way to tackle this problem using different operations and parameters.

The most interesting qualities that make MHs so interesting from an application point of view are [188]:

- Simplicity: their algorithms can be usually summarized in a few lines of pseudo-code and their implementation is therefore straightforward;
- Flexibility: they can treat any problem as a “black-box” optimization problem. There is no need for information regarding the function like its gradient or Hessian. This allows these methods to be used in a very wide variety of problems, in which more classical approaches fail;
- Ergodicity: they contain mechanisms, usually based on randomization techniques or statistical models, to escape local minima and search into highly multimodal functions.

Some possible ways to classify MHs have been proposed by Blum and Roli [16]. The most interesting one is based on the number of solutions used at the same time. Some methods (e.g. Tabu Search [48]) work on a single solution at a time while others (such as Bio-Inspired Algorithms methods) use a population of solutions. This aspect not only allows them to cover the search space with more points, but the elements of the population can interact with one another in order to improve the behavior of the search algorithm.

1.1.3 Bio-Inspired Algorithms

If you think evolution means you're a random accident,
you don't understand evolution (or accidents).

– Richard Dawkins

Bio-Inspired Algorithms (BIAs) [188] use strategies observed in nature (such as evolution, cooperation, social behavior, emergence) to drive the generation of novel solutions starting from a population of randomly generated ones. The first implementations of these techniques for global optimization were developed in the mid-seventies with the pioneering work of Holland on Genetic Algorithms (GA) [56] and of Rechenberg on Evolution Strategies (ES) [142]. They employed the theories developed by Darwin ("survival of the fittest") to generate better elements (called "individuals", to mimic the nature-inspired concept) in the population. The main idea behind this is to treat the function to be optimized as the "fitness" of the individual: an individual that obtains a good result, or that has a good fitness, has more probabilities to survive and to pass its genes to the next generation. New individuals are created by the existing ones by the mathematical equivalents of the biological crossover and mutation. The first one generates a "child" by merging the features of two "parents", the second one alters one or more values in an individual to increase diversity within the population. These operations are repeated through the generations, and the population improves its average fitness leading to better results. The general structure of an Evolutionary Algorithm (EA), one of the main examples of BIAs, is shown in Algorithm 1.

The terminology and taxonomies reported in the literature about this field are not consistent. Some consider EAs as a part of BIAs; some consider EAs and BIAs as two separate families of population-based metaheuristics. From a practical point of view, the differences between these techniques are so thin [22] that, in the rest of this dissertation, the terms "evolutionary algorithms", "bio-inspired algorithms", "population-based metaheuristics" will be simply considered as equivalent (although, in general, this is improper [93]), since most of the approaches, experiments and conclusions drawn in the next chapters are valid for all of them.

Algorithm 1 Basic structure of an evolutionary algorithm. Bio-inspired algorithms share the same paradigm.

```
$P \leftarrow InitializePopulation()$
$Evaluate(P)$
**while** Termination Criteria not met **do**
    $P \leftarrow UpdatePopulation(P)$
    $Evaluate(P)$
    $P \leftarrow SelectNewPopulation(P)$
**end while**
```

This field has been subject to a great expansion in the nineties, when many successful techniques have been proposed such as Particle Swarm Optimization [73], Differential Evolution [158], Ant Colony Optimization [156][1].

Despite the lack of full theoretical proofs (although several works are trying to fill this gap [5]), they proved to obtain outstanding performance in many situations (e.g. uncertain environments [67]) and in many applications such as civil engineering [74], economics [147], electric power systems [2], materials design [133], medical image processing [100, 107], networking design [36], robotics [119], scheduling [52], sustainable energy development [196], timetabling [85], wireless sensor networks [81], and many others.

These two aspects (lack of theoretical proofs and good performance under different conditions) point to another aspect of these algorithms: they are usually quite sensitive to the tuning of their parameters. This tuning is usually performed by using trial-and-error approaches or by testing a grid of possible combinations of parameters, but many more systematic (as well as scientific) approaches have been presented [39].

This thesis will deal with two of the most successful bio-inspired optimization techniques, Particle Swarm Optimization and Differential Evolution, but the conclu-

[1]More recently, a huge number of algorithms with an alleged biological or natural inspiration, from frogs to cuckoos to jazz musicians, have been proposed. Please refer to [157] for a critical analysis and to [181] for a specific example.

sions can be easily extended to other metaheuristics.

Particle Swarm Optimization

Particle Swarm Optimization (PSO) is a bio-inspired algorithm introduced by Kennedy and Eberhart in 1995 [73]. It is based on the simulation of the social behavior of bird flocks looking for food. Especially thanks to its simple implementation and ability to reach good results quickly when compared to similar population-based techniques, since its introduction PSO has been applied to a very wide variety of problems [136] and many variants of the original algorithm have been proposed [23].

During PSO's execution a population of P elements (commonly called "particles") moves within a function domain searching for the optimum of the function (best fitness value). The motion of the i^{th} ($i = 1, \dots, P$) particle can be described by the following two simple equations which regulate its velocity and position:

$$\begin{aligned}
\vec{v}_i(t) &= w \cdot \vec{v}_i(t-1) \\
&+ c_1 \cdot rand() \cdot (\vec{BP}_i - \vec{P}_i(t-1)) \\
&+ c_2 \cdot rand() \cdot (\vec{BGP}_i - \vec{P}_i(t-1)) \\
\vec{P}_i(t) &= \vec{P}_i(t-1) + \vec{v}_i(t)
\end{aligned}$$

where $v_i\vec{(t)}$ and $P_i\vec{(t)}$ are respectively the velocity and position of the i^{th} particle at time t; c_1, c_2 are constants that represent how strongly cognitive and social information affects the behavior of the particle; w is an inertia factor (that sometimes depends on time t [151]); $rand()$ returns random values uniformly distributed in $[0, 1]$, and BP_i is the best-fitness position visited so far by particle i.

In the basic *global-best* PSO algorithm, $\vec{BGP}$ is the best-fitness position visited so far by any particle of the swarm, therefore it is the same for all particles ($\vec{BGP}_i = \vec{BGP}$, $\forall i$). In several variants, called *local-best* PSOs, the swarm is subdivided into smaller neighborhoods which can assume different topologies [23]. Figure 1.3 shows some of the most common choices. The one on the left is the *global* one, in which each particle is connected to all others. In the second one, called *ring*, the neighborhood of each particle is composed by itself, the previous K and next K

particles, where K indicates the radius of the neighborhood. In the one on the right (*star*), one particle is selected as "center" and acts like in a "global-best" PSO, while the neighborhood of the other particles is limited to the central one and themselves. The impact of the communication between particles on the performance of PSO is still an open research topic [40].

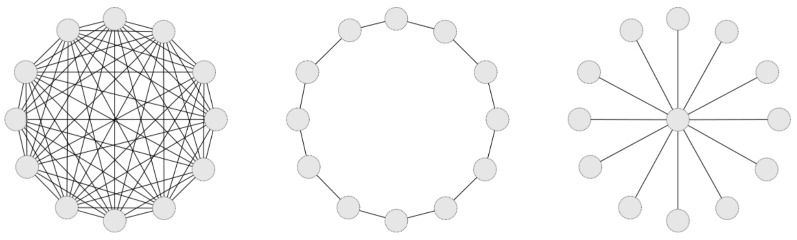


Figure 1.3: Three commonly used PSO topologies used in this work: global, ring ($K = 1$, since each particle has only one neighbor on the left and one on the right) and star.

The number of proposals of PSO variants are countless [9]; among the most interesting approaches can be considered:

- several topologies have been proposed in the years [175]. Two interesting variations are (i) the ones that use dynamic topologies which vary randomly [121] or according to the results of the optimization process [148] and (ii) the ones that rely on sub-swarms that evolve partially independently of one another and share little information to preserve diversity within the population [89];

- hybridization consists of the insertion of operators taken from other methods in the original algorithm. PSO has been successfully hybridized with other MHs like genetic algorithms [70, 149, 150] or cuckoo search [179], mathematical methods such as branch and bound [46], and local search methods [112, 140, 66, 195];

- asynchronous versions [141, 176], in which each particle does not need to wait for the other ones before updating its position, but does so it as soon as its evaluation is finished. This approach proved to be successful especially for parallel implementations [75, 115];

- extension to multi-objective [13] and dynamic problems in which the landscape changes during optimization [50].

Differential Evolution

Differential Evolution, first introduced by Storn and Price [158], has proved to be one of the most successful EAs for GCO, especially when the function to be optimized is multi-modal and non-separable [177]. DE follows a simple strategy in which the individuals of the current generation are perturbed by the scaled differences of other randomly-selected and distinct individuals. Therefore, no separate probability distribution has to be used for generating the offspring [28]. This way, in the first iterations the population members are widely scattered in the search space and possess great exploration ability. During optimization, the individuals tend to concentrate in the regions of the search space with better values, so the search automatically exploit the most promising areas [120] self-adapting its exploitation-exploration balance during evolution.

In DE, new individuals that will be part of the next generation are created by combining individuals of the current one. Every individual acts as a parent vector and, for each of them, a donor vector is created. In the basic version of DE, the donor vector for the i^{th} parent ($\vec{X_i}$) is generated by combining three random and distinct individuals $\vec{X_{r1}}$, $\vec{X_{r2}}$ and $\vec{X_{r3}}$. The donor vector $\vec{V_i}$ is calculated by what is called an *adaptive-mutation* of difference vectors as follows:

$$\vec{V_i} = \vec{X_{r1}} + F \cdot (\vec{X_{r2}} - \vec{X_{r3}})$$

where F (scale factor) is a parameter that strongly influences DE's performances and typically lies in the interval $[0.4,\ 1]$. In the last few years, several DE mutation strategies have been proposed, experimenting with different base vectors and different numbers of vectors used as perturbations. They are commonly recognized by a simple nomenclature: DE/{operator}/{number of difference vectors}. For instance, the original method described above is called DE/rand/1, which means that the first element of the donor vector equation $\vec{X_{r1}}$ is randomly chosen and only one difference vector (in our case $\vec{X_{r2}} - \vec{X_{r3}}$) is added. Table 1.1 shows some of the most common choices.

Table 1.1: DE Mutation Strategies. $\vec{X_{best}}$ represents the individual with the best fitness in the current population.

Strategy	Formula
DE/best/1	$\vec{V_i} = \vec{X_{best}} + F \cdot (\vec{X_{r1}} - \vec{X_{r2}})$
DE/target-to-best/1	$\vec{V_i} = \vec{X_i} + F \cdot (\vec{X_{best}} - \vec{X_i}) + F \cdot (\vec{X_{r1}} - \vec{X_{r2}})$
DE/best/2	$\vec{V_i} = \vec{X_{best}} + F \cdot (\vec{X_{r1}} - \vec{X_{r2}}) + F \cdot (\vec{X_{r3}} - \vec{X_{r4}})$
DE/rand/2	$\vec{V_i} = \vec{X_{r1}} + F \cdot (\vec{X_{r2}} - \vec{X_{r3}}) + F \cdot (\vec{X_{r4}} - \vec{X_{r5}})$
DE/rand-to-best/2	$\vec{V_i} = \vec{X_{r1}} + F \cdot (\vec{X_{best}} - \vec{X_i}) + F \cdot (\vec{X_{r2}} - \vec{X_{r3}}) + F \cdot (\vec{X_{r4}} - \vec{X_{r5}})$

After mutation, every parent-donor pair generates a child ($\vec{U_i}$), called trial vector, by means of a crossover operation. The two most common crossover strategies employed are called *binomial* (or uniform) and *exponential*. The former follows this simple strategy:

$$U_{i,j} = \begin{cases} V_{i,j} & \text{if } (rand_{i,j} \leq CR \text{ or } j = j_{rand}) \\ X_{i,j} & \text{otherwise} \end{cases}$$

The j^{th} component of the i^{th} donor vector is obtained by means of uniform (or binomial) crossover, where $rand_{i,j}$ is a uniformly distributed random number in the range $[0,1]$, CR is the crossover rate, and j_{rand} is a randomly selected dimension.

In *exponential* crossover, instead, an integer L is first computed following this pseudo-code:

```
$L = 0$
do
$L = L + 1$
while $((rand_{i,j} \leq CR)$ and $(L \leq D))$
```

where D is the dimension of the individuals. Then, the trial vector is finally generated:

$$U_{i,j} = \begin{cases} V_{i,j} & \text{for } j = \langle j_{rand} \rangle_D \ldots \langle j_{rand} + L - 1 \rangle_D \\ X_{i,j} & \text{otherwise} \end{cases}$$

where $\langle \cdot \rangle_D$ represents the *modulo D* function.

After the crossover operation, the newly-generated individual $\vec{U_i}$ is evaluated and its fitness $F(\vec{U_i})$ is compared with the one obtained by its parent $F(\vec{X_i})$. Following an elitist strategy, if $F(\vec{U_i}) \leq F(\vec{X_i})$, the newly generated offspring replaces its parent in the population, otherwise the parent "survives" and will be part of the next generation.

Due to its simplicity and effectiveness, many researchers have proposed modifications to DE's basic algorithm [28, 120]. Among the many variants of DE operators, one of the most successful ideas was to adapt the strategy (mutation, crossover, and parameter values [18]) during optimization, according to the results obtained. EPSDE [97] starts with a pool of mutation and crossover strategies; at first, each individual is randomly associated with a strategy from the pool and uses it to generate a trial vector. If the trial vector is better, it keeps the parent's strategy, otherwise a new strategy is assigned to the parent. Other strategies that employ a pool of possible solutions are Composite DE [180], and Strategy Adaptation DE [139].

Like PSO, hybridization of DE with MHs [91, 111, 194] as well as with local search [122, 189] proved to be successful. Variants of DE have been successfully proposed for discrete [132] and binary [131] problems, multi-objective optimization [7, 187, 143] and dynamic environments [103].

1.1.4 Multi-Objective Optimization

Before moving forward to the next topic, let us go back to the first scenario in the beginning of Section 1.1 and suppose that the designer has also to minimize the overall production cost of the fan. In general, more than two objectives can be optimized at the same time. The simplest strategy to tackle a situation like this consists of merging the objectives $f_j, j = 1, \ldots, N$ into a single metric by a weighted sum:

$$f(\vec{x}) = \sum_{j=1}^{N} w_j \cdot f_j(\vec{x})$$

where $w_j \in \mathbb{R}$ is a weight associated to objective j. This strategy has several drawbacks, the main being (i) the strong impact that the choice of the weights have on the final result and (ii) the different measurement units by which objectives may be measured which makes summing them senseless. A better strategy, especially when the different goals are in conflict with each other, is the one known as Pareto Optimization. The goal of this strategy is not to find a single solution, but an entire set of non-dominated solutions. A solution is said to be non-dominated if all other solutions are worse on at least one of the objectives. The collection of non-dominated solutions represents the approximation of the Pareto Front (PF). Figure 1.4 shows a simple two-dimensional PF. It can be said that, moving through the PF, it is possible to find different trade-offs between the conflicting goals.

MHs have proven to be very successful in tackling multi-objective problems [24, 198]. One of the most common techniques is an extension of classical genetic algorithms, called Non-dominated Sorting Genetic Algorithm (NSGA-II) [32].

Non-dominated Sorting Genetic Algorithm

Non-dominated Sorting Genetic Algorithm (NSGA-II) is one of the most used multi-objective optimization method, introduced by Deb et al [32]. NSGA-II obtains good results, particularly in solving non-convex and non-smooth problems, thanks to an elitist strategy in which the elements that survive in the next generations are selected according to their *rank* and their crowding distance, respectively defined by:

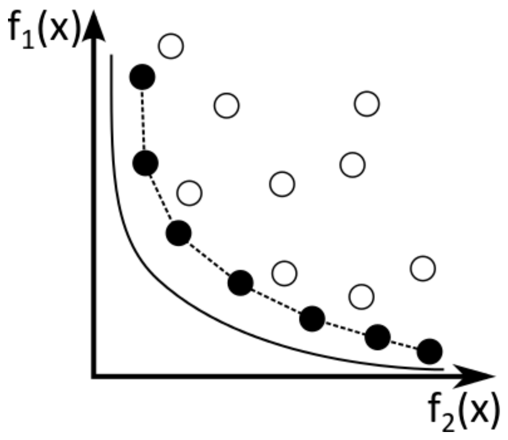


Figure 1.4: Example of a Pareto Front for a minimization problem. $f1$ and $f2$ are the two objective functions. The solid black line is the Pareto Front, the dotted line is its approximation obtained by the non-dominated solutions (black dots). The white dots represent dominated solutions. The goal of an optimization algorithm is to move the approximation as close as possible to the real Pareto Front.

- a fast non-dominated sorting procedure (see Algorithm 2), which is a procedure that assigns a *rank* to each particle $\vec{P}_i$, i.e. the number of particles that dominate $\vec{P}_i$: non-dominated particles have $rank = 0$, and so on. This algorithm is composed of two phases: in the first one (lines 1 to 16) it computes the number of individuals that dominate each particle in the population, so that the non-dominated ones have $rank = 0$. All the elements with $rank = 0$ compose the front F_0. In the second phase (lines 17 to 31) the *ranks* are assigned to the dominated particles: the elements that are dominated only by individuals belonging to F_0 have $rank = 1$ and compose F_1, and the same operation is repeated until the *rank* has been assigned to all particles;

- a crowding distance that guarantees diversity between solutions without the need of any user-requested parameter. The algorithm for crowding distance computation is presented in Algorithm 3. For each element the crowding distance is initialized to zero, then for each goal g it is incremented by the (normalized) distance between the two elements that precede and follow it according to g. When an element is the best (or the worst) according to a goal its distance is set to ∞ so that it is always chosen among the particles with the same *rank*. The same algorithm can be applied to parameter values instead of goals.

Algorithm 2 Fast Non-dominated Sorting algorithm. The output is the *rank* for each individual in the population.

```
1:  for all $\vec{P_i} \in Population$ do
2:      $S_i \leftarrow \emptyset$
3:      $n_i \leftarrow 0$
4:      for all $\vec{P_j} \in Population$ do
5:          if $\vec{P_i}$ dominates $\vec{P_j}$ then
6:              $S_i \leftarrow S_i \cup \vec{P_j}$
7:          end if
8:          if $\vec{P_j}$ dominates $\vec{P_i}$ then
9:              $n_i \leftarrow n_i + 1$
10:         end if
11:     end for
12:     if $n_{\vec{P_i}} = 0$ then
13:         $\vec{P}_{i\,rank} \leftarrow 0$
14:         $F_0 \leftarrow F_0 \cup \vec{P_i}$
15:     end if
16: end for
17: $i \leftarrow 0$
18: while $F_i \neq \emptyset$ do
19:     $N \leftarrow \emptyset$
20:     for all $\vec{P_i} \in F_i$ do
21:         for all $\vec{P_j} \in S_i$ do
22:             $n_j \leftarrow n_j - 1$
23:             if $n_j = 0$ then
24:                 $\vec{P}_{j\,rank} \leftarrow i + 1$
25:                 $N \leftarrow N \cup \vec{P_j}$
26:             end if
27:         end for
28:     end for
29:     $i \leftarrow i + 1$
30:     $F_i \leftarrow N$
31: end while
```

The NSGA-II algorithm is basically derived from a classic GA. First, a population of P individuals is randomly initialized. Then, in each generation, some elements are selected into a mating pool and are subject to crossover and mutation. These offsprings and the current population are then merged, and the best P elements are passed to the next generation, in an elitist way, according to the non-dominated sorting. In case two individuals have the same *rank*, the one with the higher crowding distance is selected; by doing so, it is possible to preserve diversity among the indi-

Algorithm 3 Crowding distance computation. In this case goals are considered, but parameter values can be used instead.

```
for all $\vec{P_i} \in Population$ do
    $\vec{P}_{idistance} = 0$
end for
for all $g \in Goals$ do
    $Population \leftarrow sort(Population, g)$
    $\vec{P}_{0distance} \leftarrow \infty$
    $\vec{P}_{Ndistance} \leftarrow \infty$
    for $i = 1 \ldots N-1$ do
        $\vec{P}_{idistance} += (f_g(\vec{P_{i+1}}) - f_g(\vec{P_{i-1}}))/(f_g{}^{max} - f_g{}^{min})$
    end for
end for
```

viduals. This distance can take into consideration the fitness or the encoding of the individuals, to increase the diversity of results or of the population, respectively.

1.2 Object Recognition

Object recognition is the task of finding a given object in an image or in a video sequence. This task can be difficult for humans and, in computer vision, this is far more challenging because of artefacts (related with the acquisition system, the environment or other optical effects like perspective, illumination changes, etc.) which may affect the aspect even of easy-to-identify and well-defined objects. These problems may cause an object to have different appearance depending on the specific conditions under which an image has been acquired. The task may become even more challenging when "interferences", such as partial occlusions or noisy backgrounds, are present. Finally, and possibly most importantly, the task is further complicated by the intrinsic variability exhibited by different instances of objects belonging to large general categories or, even more relevantly, by living beings, or parts of their bodies.

In many real-world applications which rely on object recognition, such as medical imaging, advanced driving assistance, image understanding, surveillance, virtual reality, some requirements need to be satisfied:

- precision in object detection;
- robustness against noise (like the alterations described in the previous paragraph) and occlusions;
- fast execution, up to real-time performances, which are required in some tasks.

All object recognition tasks, implicitly or explicitly, must rely on a model of the object that is to be detected. This is reflected in the main computer vision approaches, which can be divided into bottom-up and top-down.

1.2.1 Bottom-Up Approaches

The more classical bottom-up (or feature-based) approaches are based on processes where all significant generic low-level features (e.g. edges, colors, textures, regions, etc.) are extracted from the image, and then "composed" into sets which may represent a target. These processes rely on a "natural" strategy which mimics biological perception, in the absence of any expectations about what the object of perception may be. In this case, knowledge about the object (its model) is only applied after an exhaustive extraction of features from the image. In practice, this usually leads to a lack of specificity, as all image regions are usually explored independently of any expected content. This may further turn into a lack of accuracy or an increase in computation burden, due to the huge number of possible outcomes that needs to be taken into consideration.

1.2.2 Top-down Approaches

On the other hand, top-down (or model-based) approaches rely on knowledge about the object to be detected, as well as about the physical laws to which both the object itself and the image acquisition process obey. A prototype model is created which describes the most important invariant features (e.g. color, shape, relative position with respect to other objects, etc.) of the object under consideration. At the same time, knowledge about the physics of the object and the imaging process makes it possible to define a transformation, from the object space to the image space, which can represent all possible poses or deformations of the object as they may appear within the image. After the acquisition, a tentative hypothesis is made about object location, deformation and orientation, which represents a point in the transformation domain. The hypothesis is then checked by evaluating the similarity between the expected appearance of the modeled object in the image and the actual content of the corresponding image region.

This approach is generally preferable when two criteria are satisfied:

- a priori knowledge is available, which can limit the size of the transformation domain;
- the implementation of the transformation is sufficiently fast to allow a search algorithm to efficiently explore such a domain.

This is a very general class of methods which can be adapted to virtually any object that contains some invariant features (like color, shape, feasible and infeasible deformations ...) that can be represented by a set of parameters, as well as of knowledge about the physical laws to which the object obeys.

The definition of the model is performed in two steps:

- selection of the most relevant features of the object, based on the observation of a significant subset of object instances;
- definition of the ranges within which the features may vary, based on the knowledge of the physical laws which regulate the object and the image acquisition process, in order to account for all possible transformations (perspective effects, other geometric transformations, deformations due to noise, etc.) by which the object may be affected.

After the definition of the model, a similarity measure must be also defined. It takes in input a possible hypothesis on the model and must reflect the similarity between the hypothesis and the object to be identified in the image, in order to reach its maximum when the image representation of the model is superimposed to the object. The problem has now become a GCO problem: figure 1.5 shows a general scheme of this process, when a MH is used to solve it.

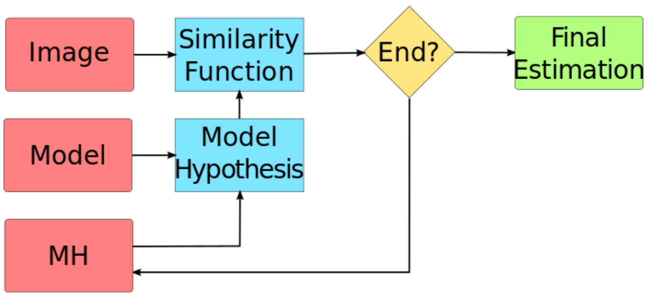


Figure 1.5: Scheme of Model-based Object Recognition. The MH generates a new hypothesis about the pose/deformation of the model. This new hypothesis is compared to the actual image using the objective (similarity) function and the process is repeated until a termination criteria (usually based on time or required precision) is met.

From a more practical point of view, “deformable models” are one of the most common implementations of model-based object recognition.

Deformable Models

Deformable models (DMs) [162, 163] are a specific template-matching [19] implementation and refer to curves or surfaces, defined within the image domain, that are deformed under the influence of “internal” forces, related with the curve features, and “external” forces, related with the image features. Internal forces enforce regularity constraints and keep the model smooth during deformation, while external forces are defined to attract the model toward the object of interest.

Parametric models are a family of DMs which represent curve (or surfaces) by means of an equation that represent their parametric forms during deformation. This allows direct interaction with the model and leads to a compact representation for a fast implementation. One of the first examples, called “snakes” or Active Contour Models, was presented in [71] and was used for image segmentation. A model is defined by n points and is deformed in order to minimize an energy function. Active Shape Models (ASMs) [26] add more prior knowledge to deformable models: they derive a “point distribution model” from sets of labeled points (landmarks) selected by an expert in a training set of images; in each image, a point is placed on the part of the object corresponding to its label. While this model has problems with unexpected shapes, since an instance of the model can only take into account deformations which appear in the training set, it is robust with respect to noise and image artefacts, like missing or damaged parts. Active Appearance Models [25] extend ASMs by considering not only the shape of the model, but also other image properties, like texture or colour. Metamorphs [61] integrate both shape and appearance in a unified space by encoding the boundaries of an object as probability maps. Finally, “deformable templates” [64] represent shapes as deformations of a given prototype or template which mathematically describes the prior knowledge about the object shape as the most likely appearance of the object. One needs to provide a mathematical description of the possible relationships between the template and all admissible object shapes. A template is specified by a set of parameters which control its deformation until it matches the target.

Optimization of deformable models can be a very hard task, especially in cluttered situations. Therefore, the use of MHs with deformable models has been successful many times in different tasks: optimization of the definition of the models [54], object detection [65], localization [124] and segmentation [105].

Chapter 2

Parameter Tuning and Meta-Optimization

Nothing is impossible for the man who doesn't have to do it himself.

– Abe H. Weiler

In their recent review on Swarm Intelligence and Bio-Inspired Computation [188], Yang et al state that "parameter tuning is a very active research area which requires more research emphasis on both theory and extensive simulations". Nevertheless, fine tuning the design of these techniques is not a simple task and must take many variables into consideration, among which the most important are:

- the nature of the problem(s) under consideration;
- the ability to generalize results over different (classes of) problems:
- the constraints of the problem, such as the restrictions imposed by computation time requirements (e.g., real-time performances);
- the quality indexes used to assess an algorithm's performance.

Problem

The features of the function to be optimized are obviously the main factor to be considered when designing a MH. It is common knowledge that a solver tailored on a specific problem has more ability to tackle it when compared with a general-purpose one. However, it is not always easy to extract the computational properties that characterize the fitness landscape [135] of a problem under consideration such as isotropy, modality, ruggedness, separability, particularly when considering real-world tasks. This lack of domain knowledge on the problem at hand can be partially overcome by a better knowledge of the algorithm used. In Bio-Inspired Optimization, this translates into the need of understanding the "meaning" of the many parameters that regulate the algorithms' behavior and how these settings influence their performance. In spite of this, MHs are often applied without taking this into account and "standard" parameter settings tend to be used, which may lead to failures even in simple tasks.

Generalization

If different problems are better tackled by different solvers, a straightforward consequence is that there exists a relationship between solvers and problems, and that similar problems are likely to require similar solvers. Consequently, it could be useful to find patterns or direct relationships between the characteristics of a solver and the problems on which it performs best. If two problems have similar computational properties, and are correctly tackled by the same solver, it should be possible to generalize this information, and use the same solver on other problems that share those properties.

An aspect that needs to be taken into consideration when evaluating MHs is that there are algorithms (or algorithm configurations) that can averagely reach good results on different problems, while others may be less robust, and exhibit performances characterized by a larger variance. The former can be used more reliably for unknown problems, but the latter can be better tailored for a specific task; this behavioral difference may be quite difficult to catch. A consequence of this statement is that, although it can be useful to find a good set of parameters for a specific MH dealing with a specific problem, in the long run it is probably more important to understand why such a set works, and how a change in its values affects the algorithm's performances.

Constraints

The approach used most frequently when evaluating a MH is simply to test it on several fitness functions and extract some aggregated information, like average or median fitness reached after a certain time (or number of fitness evaluations) in several independent runs. A big issue with this way of testing is that the conditions under which experiments are performed have a very strong influence on the results. Consider the plots in figure 2.1: on the x axis there is the time spent optimizing the function $f(x)$, whose best value reached at a certain time t is plotted on the y axis. Let us suppose to have two algorithms A and B. If they are let to run for t_1 seconds, we would say that the first performs better than the second one but, if we wait until t_2, the opposite appears to be true. This shows that choosing the better performer requires a precise definition of the conditions under which the comparison is made: a statement like "Algorithm A performs better than B" makes no sense unless this specification is also clearly stated.

Quality indexes

Usually, the main property investigated in analyzing algorithms' performance is their ability to reach the best fitness value. However, in many real-world applications, this is not the only goal to be aimed at. Let us consider some different real-world optimization tasks:

- when dealing with object tracking and recognition in images and videos [174], a solution must be found quickly, and it is not necessary to reach the global optimum but a point which is "close enough" is sufficient; therefore, an algorithm that is able to consistently reach good results (although not perfect) in a short time is to be preferred to another one that reaches better results, but takes much longer to do so;

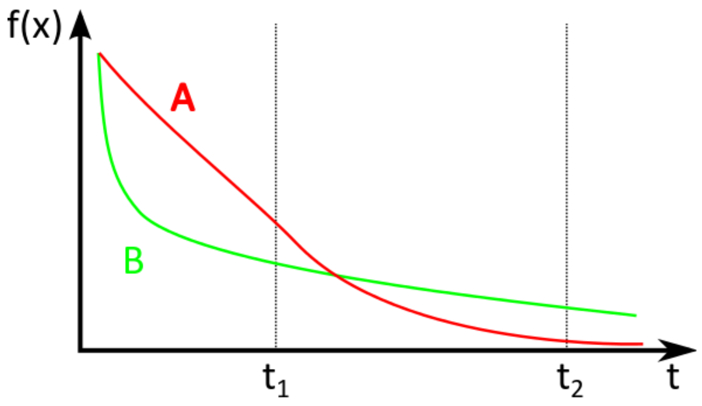


Figure 2.1: Comparing the performance of two optimization algorithms A and B. Time is on the x axis, objective function value on the y. If the algorithms are stopped when $t = t_1$, A is better than B, but if the time budget is increased up to $t = t_2$, the result is the opposite. Therefore the question is: which algorithm is better?

- on the contrary, in other tasks such as structural design [74], the problem is entirely different because there are virtually no time constraints; then, it is possible to run many experiments and keep only the best solution ever found. Some of the properties that were desirable in the previous task, such as reaching good results quickly, are not relevant in this problem, and an algorithm that keeps refining the solution over time, although very slowly, is preferable.

Of course, the ideal case would be obtaining the best possible results in the shortest possible time; it is easy to understand that, most often, these two goals are conflicting with each other, and a trade-off is needed.

2.1 Parameter Tuning

With these considerations in mind, it is very difficult to evaluate the performance of an algorithm and choose a good set of parameters for a given problem. As a consequence, comparing different algorithms is also very complicated because, for a fair comparison, each algorithm must be used “at its best”. This does not always happen; in fact, in the literature, many examples can be found where it is evident that the effort spent by the authors for tuning and optimizing the method they propose is much larger than the effort spent for tuning the ones used as reference in the comparison. This may obviously lead to erroneous interpretations of the results and to wrong conclusions.

Focusing more on MHs, several methods have been proposed to deal with their sensitivity to their parameters. These methods can be grouped into two main classes:

- Parameter tuning [58]: the parameter values are chosen off-line and they do not change during evolution, which is the case on which this work will focus;

- Parameter control [38]: the parameter values may vary as the optimization proceeds, according to a strategy that depends on the results of the process. These changes can be driven either by fitness improvement (or by its lack) or by some properties of the evolving population, like diversity or entropy.

MH parameters can be grouped according to two main types:

- Numerical parameters: either integers (e.g., number of population members) or real numbers (e.g., DE crossover rate);

- Nominal parameters (also called categorical parameters, or settings): these parameters represent different design choices (e.g., different PSO topologies) and cannot be ordered reasonably; as a consequence, they are not searchable by continuous optimization methods.

Unfortunately, a very common method to tune MH parameters is a "manual" search based on a trial-and-error approach, in which the developer first tests a parameter set and, driven by its results, tweaks some parameters repeating this operation until results become good enough, or he/she gets tired. Another approach is to test a representative grid of parameter sets [21, 44, 87], but it is a complex task, mostly because the parameters' search space can be very large, and may become huge when considering all the possible issues discussed before.

Trial-and-error approaches can be considered “manual versions” of how bio-inspired techniques work: they generate possible solutions and, based on the results obtained, explore different areas of the search space until a stopping criterion is reached. At the same time, finding good parameters is a multi-dimensional, “noisy” problem, i.e., the kind of problems in which these techniques typically perform well. Therefore, a natural way of tackling this problem could be to let a MH search the space of the algorithm’s parameters to be tuned. This approach is usually called Meta-Optimization and is summarized in Figure 2.2. According to the nature of the Tuner MH, Meta-Optimization can be referred to using other names such as Meta-EA (when an EA is used), Meta-GA and so forth.

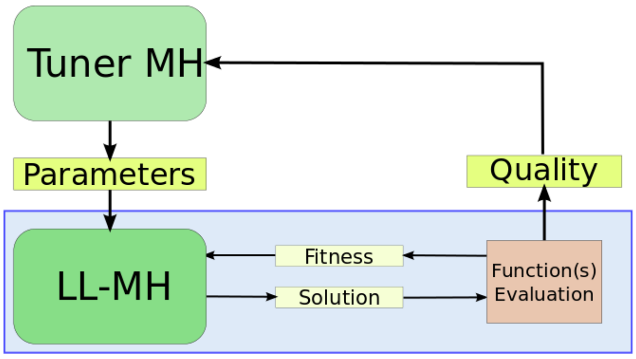


Figure 2.2: Scheme of Meta-Optimization. The Tuner MH searches in the space of LL-MH (lower level MH) parameters. When a LL-MH configuration is generated, it is tested T times and an aggregated result is the fitness of the corresponding Tuner MH individual.

The block in the lower part of the image represents a traditional optimization problem: a MH, referred to as Lower-Level MH (LL-MH) optimizes a function following such an algorithm. The Tuner MH (above in the figure) works in a very similar way, except that it does not operate in the search space of the problem to be solved, but in the search space of the parameters of LL-MH. This means that Tuner MH generates possible LL-MH configurations. For each set of parameters, an entire optimization process using LL-MH is repeated T times on the function(s) taken into consideration. An aggregated measure of the T results (e.g. the average final fitness, or the time to reach a solution), called “Quality” in the image, represents the fitness value of the individual corresponding to the configuration that has been tested of the Tuner MH population, which runs until it converges or it runs out of time.

2.2 Automatic Parameter Tuning

Several strategies have been proposed to automatically find good parameters [39, 92] (or, in a broader sense, to choose good heuristics). In this section, the most interesting methods for off-line tuning of MHs will be briefly presented. This section is not meant to offer a complete coverage of all possible approaches, but to provide the reader with insights of possible solutions to this hard problem.

2.2.1 Meta-Optimization

The idea underlying Meta-Optimization was first introduced in 1978 by Mercer and Sampson [104], while Grefenstette [51] conducted more extensive experiments with a meta-GA and showed its effectiveness, despite the limited amount of computational power available in those years. Gratch et al [49] introduced a method based on hill climbing named Composer, Freisleben et al [43] used a GA to optimize a population of underlying GAs, Bäck [8] proposed a meta-algorithm that combined principles of ES and GA to tune both continuous and discrete parameters of a GA at the same time.

In the last few years several methods using the same paradigm have been proposed: one of the most successful is called REVAC (Relevance Estimation and VAlue Calibration [116]), a method inspired by the Estimation of Distribution Algorithm (EDA [83]), in which the population explores the search space according to the probability density function of the most promising areas of the fitness landscape. In [153], REVAC proved its effectiveness by finding parameters that improved the performance of the winner of the competition on the CEC 2005 test-suite [159]. Meissner et al [102] used PSO to tune itself and applied the optimal parameter sets to neural network training; Pedersen [134] used a simple metaheuristic called Local Unimodal

Sampling to tune DE and PSO parameters, obtaining good performance while discovering unexpectedly good parameter settings.

ParamILS [62] is a method that performs a local search starting from a default parameter configuration, which is iteratively improved by modifying one parameter at a time; FocusedILS [63] improves ParamILS by spending more budget on the most promising areas. GGA [3] uses a genetic algorithm variant used to configure solvers for combinatorial problems. Luke and Talukder [95] used a Meta-EA in a massively parallel system as an optimization method: meta-evolution is not performed to find the best possible parameters of an underlying EA (GA, ES or DE), but to generate good optimizers on-line and solve the problem under consideration directly.

Multi-Objective Meta-Optimization

Multi-Objective Meta-Optimization extends the meta-optimization process by choosing the best MHs considering more goals at the same time. The first extension of Meta-Optimization to the multi-objective paradigm was proposed by Dréo [35] who used NSGA-II to optimize speed and precision of four different algorithms: Simulated Annealing, an EA and two versions of EDA. However, he took into consideration only one parameter at one time, so this approach cannot be considered a full parameter set optimization algorithm. A similar method has been proposed in [155]. The authors describe a variation of a MOEA called Multi-Function Evolutionary Tuning Algorithm (M-FETA), where the performance of a GA on two different functions represent the different goals that the MOEA has to optimize; the goal is to discriminate algorithms that perform well on a single function from those that do on more than one, respectively called specialists and generalists, following the terminology introduced in [154].

Finally, an interesting technique has been proposed by [17], in order to identify the best parameter settings for all possible computational budgets (i.e. number of fitness evaluations) up to a specified maximum. This is obtained using a MOEA in which the fitness of an element is composed of the fitness values obtained in every generation. In this way, it is possible to obtain a family of parameter sets which guarantee that the best results are reached when different computational budgets are allowed.

2.2.2 Model-Based Approaches

Other approaches for parameter tuning are model-based. Their goal is to generate meta-models that estimate the utility of a parameter by analyzing the meta-fitness landscape. A simple example is presented in [27], in which the parameters of a GA are subject to a “statistical exploratory analysis” to clarify the relationship between the parameters and performance of the algorithm. *Calibra* [1] employs factorial design [160] to extract good parameters sets, which are then refined using a local search method.

One of the most successful model-based approaches is Sequential Parameter Optimization (SPO), proposed in [12]: it is an iterative process in which, at each repetition, a set of parameter configurations is generated and their utilities are estimated according to a model. The most promising ones are then evaluated and the model is updated accordingly. In this way, it is possible not only to find good parameter values, but also to predict a range of validity for each parameter. In [138], an interesting discussion about how the effort should be divided between the initial creation of the model and its evaluation/improvement is presented.

2.2.3 Portfolio Solvers

A different approach is the one employed by Portfolio Solvers [86]. This paradigm, more frequently employed in combinatorial than in continuous optimization, consists of having a portfolio of solvers (usually trained offline [186]), each one with different properties and more capable of dealing with certain problems than the others. Then, when a new problem instance is to be tackled, they all start and only the most promising ones are allowed to continue the optimization.

2.2.4 Racing

The goal of racing algorithms is to look for the best parameters with the lowest possible number of tests. To do so, the tuner generates a population of possible configurations based on a pre-defined distribution; elements from this population are then tested and possibly discarded as soon as a statistical test indicates that there is at least one other element in the population which outclasses them; these operations are repeated until a set of good configurations is obtained. The first work using this idea is [98], and one of the most successful algorithms is *irace* [94], which has been employed in combinatorial [15], continuous [90], and multi-objective [14] optimization with good success.

2.2.5 GP-based Hyper-heuristics

These approaches work at a higher level than typical parameter tuning methods. They use Genetic Programming (GP) [76] (an evolutionary paradigm that evolve computer programs) or other techniques to generate novel heuristics using the combination of different operators and/or simpler heuristics, as well as tuning parameters [20]. Woodward and Swan proposed an "algorithmic tuning" opposed to the classical parameter tuning and used GP to generate novel selection [184] and mutation [185] operators for genetic algorithms while Oltean [127] used GP to evolve complete evolutionary algorithms.

2.3 An intermission on the comparison of metaheuristics

Before moving on with the presentation of the methods developed in this work, it is maybe useful to spend some time discussing a crucial question: what is the correct way to compare different MHs? As stated before, a sentence like "Algorithm A is better than B" is usually pointless unless precise test conditions are also provided. Therefore, it is necessary to perform tests that are able to support any conclusions of this kind and, consequently, draw only the conclusions that can be derived from the tests used. One of the most important aspects to keep in mind when working with stochastic algorithms is that any kind of supremacy hypothesis must pass a statistical test [34] to avoid the possibility that the difference is only due to the stochasticity involved in the process. Another good practice is to establish in advance the kind of tests, proceedings and evaluations to which all the MHs under study will be subject to in order to avoid to favor an algorithm, even unintentionally.

In [68] Johnson state that an experimental paper may be classified in four main categories:

- *Application Paper*: in which the purpose is to describe the impact of the algorithm on a particular application;
- *Horse Race Paper*: to prove that the algorithm is better than the state-of-the-art;
- *Experimental Analysis Paper*: to better understand the strengths and weaknesses of the algorithm;
- *Experimental Average-Case Paper*: to generate conjectures about the average-case behavior of the algorithm when direct analysis is too hard.

The methods presented in the next Chapters can be effectively employed in all these four cases:

- *Application Paper*: an automatic parameter tuning can be employed to allow the algorithm to run "at its best";
- *Horse Race Paper*: the "race" is obviously fairer if the newly-proposed algorithm and the reference ones are subject to the same parameter tuning process;
- *Experimental Analysis Paper*: the multi-objective version presented in Chapter 4 is perfectly suited for this task, as will be shown;
- *Experimental Average-Case Paper*: automatic parameter tuning methods are able to find good parameters and their ranges of validity.

The next two Chapters will present the two versions of the parameter tuning method implemented in this work, along with some tests performed on numerical problems in different conditions, and the results obtained.

Chapter 3

SEPaT

There's no such thing as simple. Simple is hard.

– Martin Scorsese

SEPaT (Simple Evolutionary Parameter Tuning) is our implementation of the general Meta-Optimization paradigm. Implementations of both Tuner-MHs and LL-MHs (see Figure 2.2) are based on DE or PSO but nothing prevents the use of other metaheuristics.

The main improvements provided by SEPaT when compared with classic Meta-Optimization approaches are:

- the way in which the elements of Tuner-MHs are compared to each other, to select which is better;
- the encoding of LL-MHs, especially as regards nominal parameters.

Comparison Method

Each individual in a Tuner-MH represents a set of parameters of the lower-level optimizer which is to be tuned. This optimizer is tested T times on a set of F functions,

$F \geq 1$. The resulting fitness is then computed based on the average and standard deviation of the best fitness obtained over all functions; that is a $2 \times F$ array of values (F averages and F standard deviations). These fitness values are then used by the tuner when its elements need to be compared. A set of parameters is better than another if it obtains better results on the majority of the F functions. In case of a tie, the winner is selected comparing the sum of Welch's t-test [146] values over the F functions:

$$R = \sum_{i=1}^{F} \frac{(\overline{X}^i{}_a - \overline{X}^i{}_b)}{\sqrt{\frac{{s^i{}_a}^2 + {s^i{}_b}^2}{T}}}$$

where a and b are the two sets of parameters to be compared, $\overline{X}^i$ and s^i are the mean and standard deviation obtained by each optimizer on function i over the T repetitions. In a minimization problem, the optimizer a is better if $R < 0$, otherwise b wins.

When it is necessary to compare more than two optimizers at a time (e.g. at the end of the optimization process to find a winner, or during optimization when the best element needs to be known for updating the elements), a full tournament is performed: each element is compared with all the others and the number of times it wins or loses is saved. The individual with the lowest number of losses is considered to be the final best. In case more than one element are as good, the ones with more wins (ties are also allowed) is considered the best. If more than one element have the same number of wins and losses, they are considered to be equivalent. By doing so the "training" functions used to tune the parameters do not need to assume values within comparable ranges, avoiding the need for normalization; nevertheless it is important that this set is balanced with respect to the properties of interest (e.g. unimodal and multimodal functions). A possible problem with this approach is that a configuration may win even if it cannot obtain good results optimizing some of the functions, since it is required only to perform better than the others on the majority of them. In this way, the parameters found by the process, despite being good on average, may be not as good on all functions. This is one of the limitations that EMOPaT (see Chapter 4) tries to overcome.

Nominal Parameters

Since the MHs used as tuners are real-valued optimization methods, the problem of representing nominal parameters needs to be addressed. The strategy followed in SEPaT consists of representing each possible choice by a vector with as many real values as the options available and selecting the option corresponding to the element with the highest value. For instance, one of the parameters that needs to be selected in PSO is topology. The possible choices are *ring*, *star* and *global* topology. This means that three values for each individual in the PSO/DE tuner are reserved to this choice and the topology corresponding to the largest one in the best particle is chosen. These elements of the individual are treated by DE or PSO just like any other element. Figure 3.1 shows the encoding of DE and PSO configurations in a Tuner MH. Please note that during the evolution of the Tuner MH all values are normalized in the range $[0,1]$ and a linear scale transformation is performed when a LL-MH is instantiated.

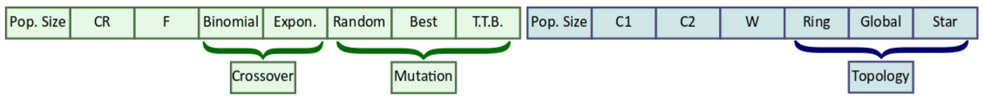


Figure 3.1: Encoding of DE (left) and PSO (right) configurations in a tuner MH.

3.1 Experiments

In this section two experiments on numerical benchmark functions using SEPaT will be presented. In the first one, the parameter sets found by SEPaT will be compared with the best ones found by a systematic search over the parameters' search space. In the second one, SEPaT's performance will be compared against *irace*, ParamILS and standard parameters over a benchmark composed of 28 functions.

3.1.1 SEPaT vs systematic search

Four classical benchmark functions having different nature (unimodal/multimodal and separable/non separable, see Table 3.1) have been selected to test SEPaT. For

Table 3.1: Benchmark functions used to test SEPaT. For every function, the table displays name, search space, formula, modality (multimodal, unimodal) and separability (separable, non separable). The fitness optima are 0 for all functions.

Name	Range	Formula	U/M	S/NS
Zakharov	$[-10,10]^n$	$\left(\sum_{i=1}^n {x_i}^2\right)+ \left(\sum_{i=1}^n 0.5\cdot i\cdot {x_i}^2\right)^2+ \left(\sum_{i=1}^n 0.5\cdot i\cdot {x_i}^2\right)^4$	U	S
Schwefel 1.2	$[-100,100]^n$	$\sum_{i=1}^n \left(\sum_{j=1}^i x_j\right)^2$	U	NS
Rastrigin	$[-5.12,5.12]^n$	$\sum_{i=1}^n \left\{x_i^2-10\cdot\cos(2\pi x_i)+10\right\}$	M	S
Rosenbrock	$[-100,100]^n$	$\sum_{i=2}^n 100(x_i-x_{i-1}^2)^2+(1-x_{i-1})^2$	M	NS

each function, the parameter search space of DE has been systematically sampled (with the sampling steps shown in Table 3.2) and for each of the 13728 parameter combinations thus generated, 10 independent repetitions generated were ran. This way, a *ground truth* has been created which can be used to assess the results of the meta-optimization. The termination criterion was set after 1000 generations and the search space size was set to 16. Figure 3.2 shows an example of results of this systematic search.

Table 3.2: Range of DE parameters allowed by the tuning algorithms. The last column shows the sampling step used in the systematic search used as reference.

Parameter	Values	Step
Crossover Rate	$[0.0, 1.0]$	0.1
Scale Factor (F)	$[0.0, 1.5]$	0.1
Population Size	[30,150]	10
Crossover	{binomial, exponential}	-
Mutation	{random, best, target-to-best (TTB)}	-

Automatic parameter tuning using SEPaT has been repeated 10 times (5 using DE as Tuner, 5 using PSO) on each function. Table 3.3 shows the parameters used

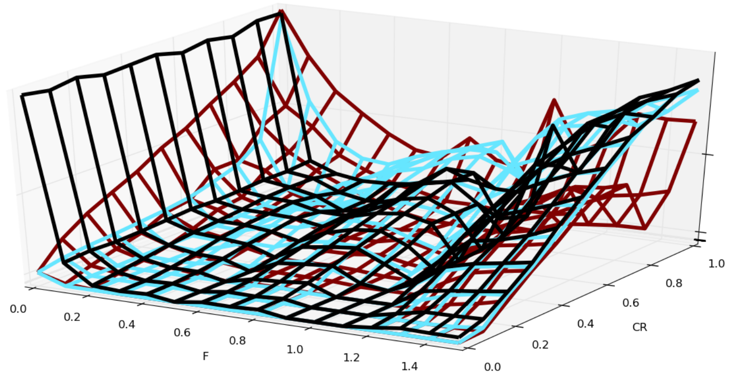


Figure 3.2: Example of results of systematic search. Scale factor F is on the x axis, crossover rate CR on the y axis and, on the z axis, the final fitness value of the Rastrigin function averaged over 10 independent runs. The three wire-frames represent the fitness values obtained using the three kinds of mutation considered (*random*, *target-to-best* and *best*).

by the tuners based on DE and PSO. In order to choose them, the tuning procedure described here has been performed over 8 benchmark functions using "standard" parameters for PSO and DE (see [173]). The best parameter configuration found, based on the number of functions for which it obtained the best fitness value, was used in our tests. This is a sub-optimal way to find good parameters for the tuners. Obviously, another meta-level to empirically find a good parameter set for the tuners could be added. This could lead to an infinite repetition of the same procedure and to an exponential increase in optimization time. However, the practical aim is just to improve the performance of manually-tuned optimizers without increasing complexity too much, when dealing with problems in which the optimizer's performance is particularly sensitive to the parameters' values, therefore this operation can be avoided.

Table 3.4 shows, for each function, the ranges of the parameters for the best 10 settings found in the systematic search and for the 10 combinations obtained by SEPaT. As can be observed, the ranges within which the parameters found by SEPaT lie are virtually overlapping with the optimal ones determined by a systematic search.

Table 3.3: Parameter values of DE- and PSO-based tuners. Notice that, with this parameter configuration, a maximum of 1536 sets of parameters (64 generations $\times$ 24 individuals) are evaluated in each run.

DE	PSO
Target-to-best Mutation	$c_1 = 1.525$
Exponential Crossover	$c_2 = 1.881$
F = 0.52	$w = 0.443$
CR = 0.91	Ring Topology
Population Size = 24, Generation = 64	

This proves that the meta-optimizer behaves correctly. An analysis of the parameter sets obtained show that $F \simeq 0.5$ is a good choice for the four functions considered, while the crossover rate needs to be high for all functions but Rastrigin, for which the best results are obtained setting a very low crossover rate.

For every set of parameters, 100 independent runs on the corresponding function have been performed. All optimizers were able to obtain a median value of 0 over the corresponding function. The Wilcoxon signed-rank test [182] confirmed the absence of statistically significant differences between the results of the systematic search and the tuning, since the p-values for the null-hypothesis “there is difference between the results obtained by using the two sets of DE parameters” were always larger than 0.01. More details on these results are presented in Table 3.5.

To reduce the computational burden, these tests have been implemented on GPU following the GPGPU (General Purpose Graphics Processing Unit) programming paradigm. Details on the library developed to perform these experiments are provided in the Appendix. In this particular task, the GPU-based version is 10.3 times faster than the single-core CPU-based one on the PC we used. Please notice that this speed-up is only indicative of this problem, since it depends on a very high number of parameters, like population size, number of evaluations, degree of parallelism and complexity of the fitness function as well as, obviously, on the hardware on which the optimizer is run.

Table 3.4: Range of the parameters found by systematic search and SEPaT. *P* is population size, *Mut* is mutation strategy, *Cross* is the crossover type.

Zakharov	Schewfel 1.2	Rastrigin	Rosenbrock
Systematic Search			
$CR \in [0.8, 0.9]$	$CR \in [0.9, 1.0]$	$CR \in [0.0, 0.4]$	$CR \in [0.8, 0.9]$
$F \in [0.5, 0.6]$	$F \in [0.5, 0.7]$	$F \in [0.1, 0.6]$	$F \in [0.6, 0.7]$
$P \in [100, 150]$	$P \in [120, 150]$	$P \in [90, 150]$	$P \in [90, 150]$
$Mut \in \{Best, TTB\}$	$Mut \in \{Best, TTB\}$	$Mut \in \{Rand, Best\}$	$Mut \in \{Best, TTB\}$
$Cross \in \{Bin, Exp\}$	$Cross \in \{Bin, Exp\}$	$Cross \in \{Bin, Exp\}$	$Cross \in \{Bin, Exp\}$
SEPaT			
$CR \in [0.799, 1.0]$	$CR \in [0.935, 1.0]$	$CR \in [0.0, 0.344]$	$CR \in [0.740, 0.939]$
$F \in [0.433, 0.714]$	$F \in [0.535, 0.667]$	$F \in [0.207, 0.596]$	$F \in [0.646, 0.686]$
$P \in [111, 146]$	$P \in [80, 150]$	$P \in [79, 149]$	$P \in [83, 150]$
$Mut \in \{Best, TTB\}$	$Mut \in \{Best, TTB\}$	$Mut \in \{Rand, Best\}$	$Mut \in \{Best, TTB\}$
$Cross \in \{Bin, Exp\}$	$Cross \in \{Bin, Exp\}$	$Cross \in \{Bin, Exp\}$	$Cross \in \{Bin, Exp\}$

Table 3.5: Comparison between the results obtained by the best parameter set found by SEPaT against the one obtained during the systematic search over the parameters' search space over 100 repetitions. The five columns report average and standard deviation obtained by the two methods, and p-value of the Wilcoxon test with the null hypothesis "There are no differences between the two distributions".

Function	SEPaT		Systematic Search		p-value
	Avg	Std. Dev	Avg	Std. Dev	
Zakharov	8.46e-05	0.95e-03	1.96e-04	2.68e-03	0.75
Schwefel 1.2	4.70e-24	1.48e-22	1.04e-21	3.28e-20	0.71
Rastrigin	1.91e-08	1.33e-07	1.99e-03	4.44e-02	0.011
Rosenbrock	0.31	1.06	0.45	2.96	0.28

3.1.2 SEPaT vs *irace* and ParamILS

The experiment presented in this section aims at proving the competitiveness of SEPaT when compared with state-of-the-art tuners. DE has been used as tuner in SEPaT to optimize PSO and DE parameter values, and compare the performance of the parameter sets obtained with the ones found by *irace* and ParamILS, along with some manually selected sets suggested by the PSO authors in [72] and in [28] for DE, to be used as "reference" (see Table 3.6). The ranges within which DE and PSO parameters were allowed to vary during the tuning phase are reported in Table 3.7.

Table 3.6: Values of DE and PSO parameters used as a reference as suggested in [72] and in [28].

Differential Evolution		Particle Swarm Optimization	
Population Size	30	Population Size	30
Crossover Rate (CR)	0.9	Inertia Factor (w)	0.721
Scale Factor (F)	0.5	c_1	1.193
Crossover	*exponential*	c_2	1.193
Mutation	*random*	Topology	*ring*

DE and PSO parameters have been tuned on a "training set" of 7 functions and a "validation set" of 21. These functions are defined in the CEC 2013 benchmark suite [88], with the only difference that the function minima have been set to 0. More details on this experiment, including the functions that compose the "training set", are provided in Table 3.8. The main goals of this experiment were to check:

1. whether SEPaT was able to get results competitive with state-of-the-art tuners such as *irace* and to improve over the standard settings;

2. whether repeatable patterns could be found in the parameter sets obtained by different techniques and by different runs of the same technique;

3. the generalization ability of every tuner.

Table 3.7: Ranges of DE and PSO parameter values allowed during the optimization. The ranges are wider than those usually considered in the literature, to allow the different tuners to “think outside the box”, and possibly find unusual parameter sets.

Differential Evolution	
Population Size	$[4, 300]$
Crossover Rate (CR)	$[0.0, 1.0]$
Scale Factor (F)	$[0.0, 2.0]$
Crossover	$\{$*binomial*, *exponential*$\}$
Mutation	$\{$*random*, *best*, *target-to-best*$\}$
Particle Swarm Optimization	
Population Size	$[4, 300]$
Inertia Factor (w)	$[-0.5, 1.5]$
c_1	$[-0.5, 4.0]$
c_2	$[-0.5, 4.0]$
Topology	$\{$*ring*, *star*, *global*$\}$

Each tuner was run seven times. The parameter sets obtained for PSO and DE are shown in Table 3.9. The ones highlighted in bold are the ones that performed best in a full tournament (see Section 4.1.2) within the seven final results. It can be seen that SEPaT and *irace* obtained similar results. For PSO (left side of the table), two main clusters can be easily recognized. The first one (SEPaT’s runs number 1, 4 and 7 and *irace*’s ones number 1, 4 and 5) has $c_1 \simeq c_2 \in [0.9, 1.7]$, $w \simeq 0.7$ and *ring* topology; the second one (SEPaT 2, 3, 5 and 6 and *irace* 3 and 7) has $c_1 > c_2$, a bigger population and *global* topology. Within the functions used in the training set the second cluster is slightly better than the first one.

Regarding DE (right side of Table 3.9), the variability of parameter values is even smaller: except for few exceptions, CR is between 0.7 and 0.9, $F \simeq 0.5$ and *exponential* crossover is selected. Mutation type is *target-to-best* or *best*; when the former is selected the number of elements is around 20, otherwise it is around 50. The variability of ParamILS parameter sets is larger in both cases and its results

Table 3.8: Comparison between SEPaT, *irace* and ParamILS. Experimental settings.

SEPaT settings
Population Size = 200, 80 Generations,
$CR = 0.91$, $F = 0.52$, target-to-best-mutation, exponential crossover
Function settings
10-dimensional Sphere, Rotated Cigar, Rosenbrock, Rotated Ackley,
Rastrigin, Composition Function 1, Composition Function 3 functions
Best fitness in 20000 evaluations averaged over 15 repetitions.

are affected by the need to discretize the search space imposed by the optimization process.

The parameter configurations of PSO and DE selected in this way have been run 200 times on the 21 functions of the CEC 2013 benchmark suite that were not used in the training phase. Their results are reported in Table 3.10 for PSO and 3.11 for DE. For each function, the method(s) that performed best according to the Wilcoxon test ($p < 0.01$) are highlighted in bold. Table 3.12 summarizes these results. It can be seen how, for both PSO and DE, SEPaT "wins" in more functions than the manually-tuned parameters used as reference, being largely the best method for tuning DE and the second after *irace* for PSO.

3.2 Discussion

In this Chapter, SEPaT has been presented and its capabilities of automatically tune parameters of metaheuristics have been successfully proven via a comparison with a systematic search and with two state-of-the-art methods, *irace* and ParamILS. SEPaT is able to find good parameters with a good repeatability and a low algorithmic complexity thanks to the properties of the bio-inspired techniques utilized. It is also able to deal with nominal parameters, an option which is usually ignored by meta-optimization techniques.

However, this approach has two main limitations:

Table 3.9: Best automatically-tuned parameter values found by SEPaT, *irace* and ParamILS in 7 runs of every tuner. The ones highlighted are the ones that obtained the best results during testing for each tuning technique.

Particle Swarm Optimization				Run	Differential Evolution			
Param.	SEPaT	*irace*	ParamILS		Param.	SEPaT	*irace*	ParamILS
c_1	1.538	1.299	2.0	1	*CR*	0.838	0.720	0.8
c_2	1.287	1.777	0.6		*F*	0.554	0.507	0.5
w	0.664	0.521	0.6		Pop. Size	51	46	166
Pop. Size	36	28	196		Crossover	exp	exp	bin
Topology	ring	ring	global		Mutation	best	best	best
c_1	2.250	1.245	**1.8**	2	*CR*	0.915	0.857	0.65
c_2	0.722	1.938	**2.0**		*F*	0.858	0.548	0.8
w	0.650	-0.207	**0.3**		Pop. Size	26	53	24
Pop. Size	50	57	**72**		Crossover	exp	exp	bin
Topology	global	global	**ring**		Mutation	TTB	best	TTB
c_1	2.167	**2.174**	2.4	3	*CR*	0.818	0.679	0.45
c_2	0.733	**0.640**	1.4		*F*	0.540	0.559	0.4
w	0.699	**0.636**	0.3		Pop. Size	52	23	252
Pop. Size	60	**63**	72		Crossover	exp	exp	bin
Topology	global	**global**	ring		Mutation	best	best	best
c_1	1.358	0.962	1.0	4	*CR*	0.863	0.784	0.5
c_2	1.153	0.914	1.6		*F*	0.547	0.862	0.8
w	0.732	0.805	0.7		Pop. Size	54	24	52
Pop. Size	22	19	84		Crossover	exp	exp	bin
Topology	ring	ring	ring		Mutation	best	TTB	best
c_1	**2.400**	1.104	1.6	5	*CR*	0.865	0.796	0.8
c_2	**0.936**	1.786	0.4		*F*	0.581	0.508	0.5
w	**0.559**	0.605	0.8		Pop. Size	53	53	180
Pop. Size	**61**	30	74		Crossover	exp	exp	bin
Topology	**global**	ring	global		Mutation	best	best	TTB
c_1	2.329	2.092	2.0	6	*CR*	**0.694**	**0.856**	**0.85**
c_2	0.887	1.208	1.6		*F*	**0.524**	**0.569**	**0.5**
w	0.649	0.572	0.3		Pop. Size	**28**	**51**	**182**
Pop. Size	62	26	52		Crossover	**exp**	**exp**	**bin**
Topology	global	ring	ring		Mutation	**TTB**	**best**	**TTB**
c_1	1.529	2.213	1.0	7	*CR*	0.160	0.751	0.3
c_2	1.827	0.788	2.0		*F*	0.499	0.564	0.3
w	0.496	0.633	0.4		Pop. Size	17	41	46
Pop. Size	56	54	52		Crossover	exp	exp	bin
Topology	ring	global	ring		Mutation	TTB	best	random

Table 3.10: PSO results obtained by automatically- and manually-tuned parameters. For each tuner, average and standard deviation of the final fitness are reported. The ones that are the statistically better than the others for the function under consideration are highlighted in bold.

Function	SEPaT		*irace*		ParamILS		Reference	
	Avg	Std. Dev	Avg	Std. Dev	Avg	Std. Dev	Avg	Std. Dev
rotated elliptic	3.75e+05	3.22e+05	**2.91e+05**	**2.98e+05**	2.09e+06	1.27e+06	7.81e+05	5.80e+05
rotated discus	**1.01e+04**	**5.51e+03**	**9.59e+03**	**4.51e+03**	1.72e+04	5.42e+03	1.47e+04	5.55e+03
powers	1.63e-01	1.74e+00	5.92e-01	4.73e+00	4.96e-05	4.52e-05	**6.80e-01**	**9.59e+00**
rotated schaffers	**3.42e+01**	**2.36e+01**	**3.12e+01**	**2.03e+01**	3.72e+01	1.31e+01	3.92e+01	1.88e+01
rotated weierstrass	**4.56e+00**	**1.39e+00**	**4.59e+00**	**1.42e+00**	5.49e+00	1.04e+00	5.52e+00	1.12e+00
rotated griewank	4.14e+00	1.08e+01	2.13e+00	4.00e+00	2.26e+00	1.51e+00	**5.27e-01**	**3.49e-01**
rotated rastrigin	**1.74e+01**	**8.37e+00**	**1.61e+01**	**7.88e+00**	2.43e+01	6.83e+00	1.84e+01	6.57e+00
rotated step rastrigin	2.61e+01	9.42e+00	**2.35e+01**	**9.37e+00**	3.16e+01	8.29e+00	2.80e+01	8.60e+00
schwefel	**2.27e+02**	**1.33e+02**	2.72e+02	1.21e+02	4.92e+02	1.66e+02	5.01e+02	1.86e+02
rotated schwefel	**9.19e+02**	**3.34e+02**	**9.17e+02**	**3.33e+02**	1.06e+03	2.00e+02	**8.72e+02**	**2.57e+02**
rotated katsuura	1.13e+00	3.02e-01	1.16e+00	2.96e-01	9.75e-01	2.50e-01	**8.31e-01**	**2.45e-01**
lunacek	**1.70e+01**	**3.52e+00**	1.96e+01	4.11e+00	2.48e+01	4.89e+00	2.37e+01	5.34e+00
rotated lunacek	**3.15e+01**	**7.57e+00**	3.52e+01	7.01e+00	4.06e+01	5.82e+00	**3.26e+01**	**6.80e+00**
griewank-rosenbrock	**7.50e-01**	**2.60e-01**	**7.72e-01**	**2.35e-01**	1.18e+00	3.32e-01	1.02e+00	3.35e-01
scaffer	3.23e+00	5.14e-01	3.26e+00	3.81e-01	3.35e+00	3.14e-01	3.25e+00	4.32e-01
CF 2	**2.93e+02**	**1.67e+02**	**3.14e+02**	**1.93e+02**	5.87e+02	1.94e+02	7.17e+02	2.30e+02
CF 4	2.17e+02	8.02e+00	**2.14e+02**	**1.47e+01**	**2.12e+02**	**1.79e+01**	2.16e+02	1.67e+01
CF 5	2.15e+02	7.34e+00	**2.13e+02**	**1.33e+01**	2.17e+02	7.18e+00	2.18e+02	8.43e+00
CF 6	1.87e+02	6.57e+01	1.67e+02	5.04e+01	**1.41e+02**	**1.76e+01**	1.58e+02	4.83e+01
CF 7	4.74e+02	8.28e+01	**4.40e+02**	**8.07e+01**	5.02e+02	6.46e+01	4.92e+02	7.68e+01
CF 8	**3.89e+02**	**1.64e+02**	**3.97e+02**	**1.73e+02**	2.62e+02	7.71e+01	**2.97e+02**	**1.05e+02**

Table 3.11: DE results obtained by automatically- and manually-tuned parameters. For each tuner, average and standard deviation of the final fitness are reported. The ones that are the statistically better than the others for the function under consideration are highlighted in bold.

Function	SEPaT		*irace*		ParamILS		Reference	
	Avg	Std. Dev	Avg	Std. Dev	Avg	Std. Dev	Avg	Std. Dev
rotated elliptic	6.61e+04	9.96e+04	3.91e+04	9.54e+04	**4.68e+03**	**4.81e+03**	1.59e+05	2.29e+05
rotated discus	1.05e+03	1.12e+03	2.73e+02	1.63e+03	**1.91e+01**	**7.61e+00**	2.37e+03	2.67e+03
powers	3.16e-08	4.36e-07	**4.28e-29**	**5.32e-28**	6.56e-05	3.20e-04	3.12e-19	1.88e-18
rotated schaffers	8.12e+00	1.04e+01	1.83e+01	1.91e+01	1.97e+00	5.12e+00	**9.77e-01**	**3.32e+00**
rotated weierstrass	5.47e+00	1.12e+00	3.57e+00	1.54e+00	3.26e+00	2.01e+00	**2.80e+00**	**1.39e+00**
rotated griewank	3.44e-01	5.12e-01	2.96e-01	1.62e+00	4.48e-01	1.64e-01	**1.42e-01**	**3.39e-01**
rotated rastrigin	**1.56e+01**	**4.31e+00**	**1.52e+01**	**7.39e+00**	2.93e+01	5.38e+00	2.08e+01	7.60e+00
rotated step rastrigin	**1.81e+01**	**6.32e+00**	2.28e+01	9.23e+00	2.92e+01	4.85e+00	2.43e+01	6.66e+00
schwefel	**6.23e+01**	**6.03e+01**	**7.10e+01**	**8.39e+01**	1.36e+03	1.81e+02	1.10e+02	5.42e+01
rotated schwefel	**1.12e+03**	**1.82e+02**	1.19e+03	3.60e+02	1.53e+03	1.82e+02	1.46e+03	1.96e+02
rotated katsuura	**1.32e+00**	**2.85e-01**	1.44e+00	2.97e-01	1.43e+00	2.77e-01	**1.39e+00**	**2.79e-01**
lunacek	**1.09e+01**	**1.49e+00**	1.28e+01	2.96e+00	3.12e+01	4.41e+00	1.14e+01	6.20e-01
rotated lunacek	**3.41e+01**	**4.93e+00**	3.71e+01	7.44e+00	4.03e+01	5.09e+00	3.90e+01	5.79e+00
griewank-rosenbrock	**5.49e-01**	**1.31e-01**	8.63e-01	1.93e-01	2.20e+00	4.35e-01	1.01e+00	1.58e-01
scaffer	3.25e+00	4.45e-01	**3.22e+00**	**5.36e-01**	**3.09e+00**	**4.59e-01**	3.22e+00	3.33e-01
CF 2	**1.48e+02**	**8.87e+01**	2.03e+02	1.34e+02	1.36e+03	2.49e+02	2.99e+02	1.59e+02
CF 4	2.06e+02	2.06e+01	2.14e+02	8.41e+00	**2.06e+02**	**1.19e+01**	2.08e+02	1.16e+01
CF 5	2.06e+02	1.61e+01	2.12e+02	1.45e+01	**2.04e+02**	**1.04e+01**	2.09e+02	1.22e+01
CF 6	**1.57e+02**	**4.58e+01**	2.03e+02	7.24e+01	1.70e+02	3.92e+01	1.87e+02	3.87e+01
CF 7	**3.50e+02**	**8.22e+01**	5.06e+02	7.39e+01	**3.41e+02**	**7.77e+01**	4.43e+02	8.97e+01
CF 8	3.26e+02	1.08e+02	3.33e+02	9.81e+01	3.06e+02	7.36e+01	2.91e+02	4.15e+01

Table 3.12: Comparison between SEPaT, *irace*, ParamILS and standard parameters. For each tuner the table reports the number of functions for which it performed better than the others. Please note that the sum is larger than 21 (number of functions) because, for each function, more than one method can be the "winner".

Method	PSO	DE
SEPaT	11	11
irace	13	4
ParamILS	2	6
Reference	6	4

- It aims at only one goal (in the cases presented here, the final fitness value): it is not possible to take multiple objectives into account, like searching for a trade-off between precision and convergence speed. Moreover, to test the same algorithm under different conditions, several independent runs of this time-expensive method must be run;
- The result consists of one set of good parameters, but the approach provides neither hints about their generality nor the reasons that caused their selection.

The next Chapter will present the multi-objective version of SEPaT, which faces these two limitations.

More details about the experiments presented in this Chapter can be found in:

- R. Ugolotti, Y.S.G. Nashed, P. Mesejo, and S. Cagnoni: "Algorithm configuration using GPU-based metaheuristics". In: Genetic and Evolutionary Computation Conference Companion (GECCO'13), pages 221-222. ACM, 2013
- R. Ugolotti, P. Mesejo, Y.S.G. Nashed, and S. Cagnoni: "GPU-Based Automatic Configuration of Differential Evolution: A Case Study". In: Progress in Artificial Intelligence, Volume 8154 of Lecture Notes in Computer Science, pages 114-125. Springer, 2013

Chapter 4

EMOPaT

No one who has ever tried it thinks conducting empirical research on heuristics is easy.

– Ronald L. Rardin and Reha Uzsoy

In this Chapter, the multi-objective extension of SEPaT, called EMOPaT (Evolutionary Multi-Objective Parameter Tuning) will be presented, along with several experiments that aim at showing the wide range of possible outcomes it may provide.

The first goal for which EMOPaT has been designed is obviously to find parameter sets that achieve good results considering the nature of the problems, more than one quality index and, more in general, the conditions under which the MH is tuned. However, it aims at the more ambitious goal of extracting information about the parameters' semantics, their role in the algorithm and the way they affect the algorithm's results and behavior by analyzing the Pareto fronts of the solutions obtained by a multi-objective MH, in this case NSGA-II. The strategy used is similar to the one presented by Deb et al [33] under the name of *innovization* (innovation through optimization). This simple yet effective paradigm aims at finding a relationship between goals and parameters (decision variables) by analyzing how these vary when moving along over the Pareto front.

From an algorithmic point of view, EMOPaT works similarly to SEPaT, the only difference being that more objectives are analyzed at the same time, following the multi-objective optimization paradigm. These objectives may take into consideration the function used, the quality indexes considered or the constraints applied, such as time constraints, problem dimensionality or others. The output of the tuning process is therefore not only a single solution as in SEPaT but an entire set of non-dominated parameter configurations, i.e., ideally, a sampling of the Pareto front for the objectives under consideration. This allows a developer to make a deeper analysis of the parameters' selection strategy. This approach can be especially relevant considering the conclusions drawn in [152]: here, Meta-EAs performed better than SPO and REVAC, but the authors pointed out the inability of Meta-EAs to provide insights about the parameters of the algorithm being tuned.

4.1 Experiments

In this section, EMOPaT will be tested in four different scenarios, to illustrate the various outcomes it can provide:

1. Firstly, it will be tested in some controlled situations, to assess its ability to produce the results it is expected to yield;

2. Then, it will be directly compared with SEPaT, to prove that it is able to reach the same results, while offering the advantages provided by the multi-objective paradigm;

3. The third experiment will show how the configurations that rely on the Pareto front can be actually considered as valid configurations, and that EMOPaT is in fact able to generate a set of solutions which represent trade-offs between the considered goals;

4. Lastly, it will be shown how it is possible to use EMOPaT to extract information on the MH under consideration that can be useful for many purposes, especially designing and comparing MHs.

4.1.1 Empirical Validation

Some examples of EMOPaT's ability to give insights about the algorithm parameters and their influence on the optimization process will be provided in this section. To do so, a number of test cases have been artificially created on which EMOPaT's performance has been assessed:

1. a numerical parameter that is useless, i.e., it does not have any influence on the algorithm's behavior;
2. a numerical parameter that is harmful, i.e., the higher its value, the worse its impact on the fitness;
3. a choice of a nominal parameter that is harmful, i.e., it constantly produces bad fitness values when selected;
4. two choices of a nominal parameter whose effects are perfectly equivalent.

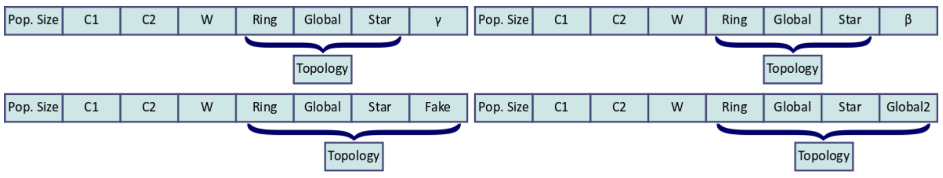


Figure 4.1: Encoding of PSO configurations in the four cases used to validate EMOPaT. From top left clockwise: useless parameter, harmful numerical parameter, equivalent and harmful topology.

A similar approach has been proposed in [110], which showed the ability of *irace*, ParamILS and REVAC to recognize an operator which was detrimental for the fitness under consideration. The results of these tests increase the confidence in the actual ability of EMOPaT to recognize the usefulness and, more in general, the role of a parameter of a MH. These tests have been performed on PSO, considering the problem of optimizing the Sphere and Rastrigin functions. The original encoding of PSO configurations (see Figure 3.1) has been modified as shown in Figure 4.1. More details on these tests are presented in Table 4.1.

Table 4.1: Empirical validation of EMOPaT. Experimental settings.

EMOPaT settings
Population Size = 64, 100 Generations,
Mutation Rate = 0.125, Crossover Rate = 0.9
Function settings
30-dimensional Sphere and Rastrigin functions
Best fitness in 20000 evaluations averaged over 15 repetitions

Useless Parameter

To perform this experiment, a parameter γ that is totally ignored by the algorithm and therefore has no effects on its behavior has been added to the PSO configurations' encoding. The goal was to analyze how EMOPaT dealt with this parameter with respect to PSO's actual parameters. Table 4.2 shows mean and standard deviation of the elements representing numerical parameters in all NSGA-II individuals (therefore, before the linear transformation, when their values are within $[0,1]$) at the end of the execution of ten independent runs. As can be observed, γ has a mean value close to 0.5 and its variance is 0.078, which is very close to $\frac{1}{12}$, the expected variance for a uniform distribution: this does not happen with the other parameters. Figure 4.2 plots the values of the Sphere function against the values of PSO parameters. While the values of the real parameters show a clear trend, the values of γ are uniformly scattered all over the allowed range. This suggests that a useless parameter can be characterized by a uniform distribution of its values. Moreover, γ shows a very low correlation with the other numerical parameters (see table 4.3), implying that its role in the algorithm is negligible.

Harmful Numerical Parameter

In this experiment, the representation of each PSO configuration has been modified with the addition of a parameter $\beta \in [0,1]$ whose only effect is to worsen the fitness proportionally to its value. At the end of each PSO fitness evaluation, the fitness f

Table 4.2: Empirical validation of EMOPaT. Useless Parameter. Mean, and variance for PSO's numerical parameters, including the useless one γ. Values of parameters are normalized between 0 and 1.

Parameter	Pop. Size	w	c_1	c_2	γ
Mean	0.159	0.555	0.586	0.332	0.441
Variance	0.0313	0.0109	0.0120	0.0103	0.0780

Table 4.3: Correlation between PSO parameter values. The ones with respect to the useless one (γ) are smaller than the ones between actual PSO parameters.

Parameter	Pop. Size	w	c_1	c_2	γ
Pop. Size	-	0.296	0.322	-0.564	-0.077
w	-	-	0.841	-0.892	-0.179
c_1	-	-	-	-0.801	-0.159
c_2	-	-	-	-	0.149

undergoes this transformation:

$$\hat{f} = (f + \beta) \cdot (1 + \beta), \quad f \geq 0$$

EMOPaT was able to constantly assign to β values close to 0 (mean $7E-4$, variance $7E-6$). This shows that the optimal value of a parameter can be identified with success. Figure 4.3 shows the values of β versus number of generations averaged over ten independent runs of EMOPaT. β starts from an average value of 0.5 (due to random initialization), but after a few iterations, its value rapidly falls to 0.

Harmful Nominal Parameter Setting

In this experiment, a "fake" fourth topology has been added to PSO configurations. When this topology is selected, it just skips the PSO process and returns a bad fitness value. The goal was to see if the selection of such a parameter would be always avoided and which values would be assumed by its corresponding element. Figure 4.4

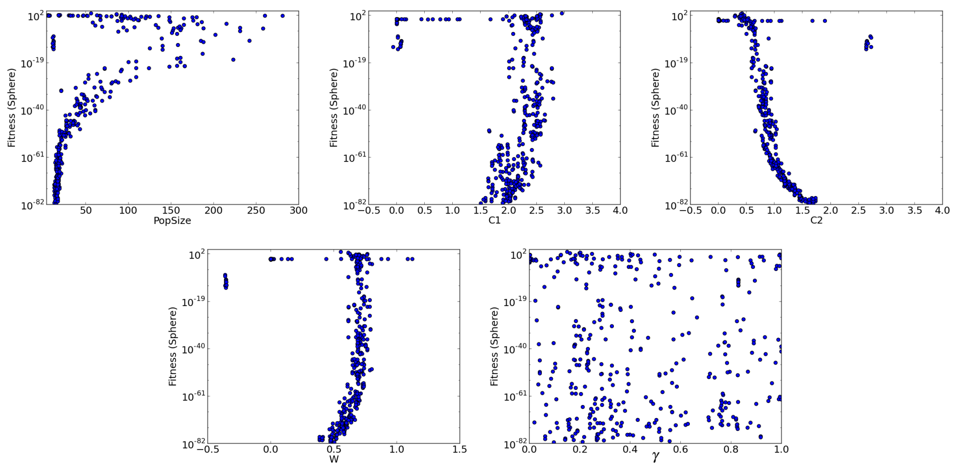


Figure 4.2: PSO parameter values versus Sphere fitness values at the end of the tuning procedure. The last graph plots values found for the useless parameter. Unlike the other parameters, that have a limited range and show a clear trend, the useless parameter γ spans across all possible values with no relationship with the fitness.

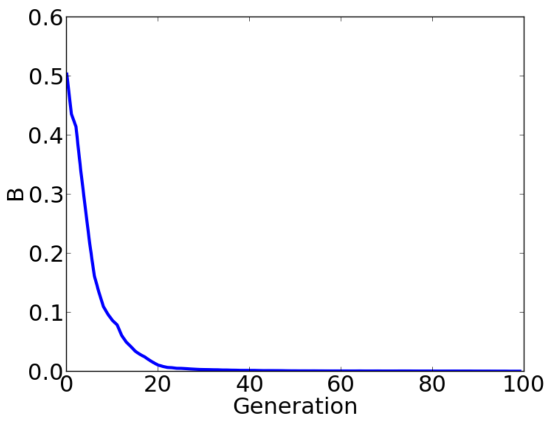


Figure 4.3: Evolution of the "bad parameter" β, averaged over all individuals in ten independent runs of EMOPaT, versus generation number.

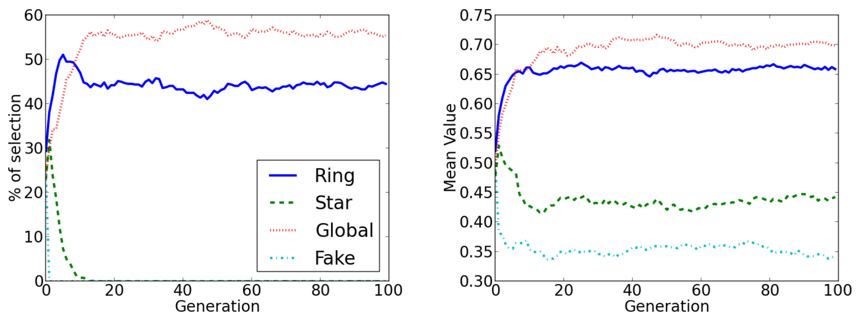


Figure 4.4: Average values assumed by the elements representing the four topologies (including the fake one) and selection percentages. On the x axis, number of NSGA-II generations. Results are averaged over 64 individuals in 10 independent runs.

shows that the fake topology is immediately discarded, and after only 2 generations is no longer selected in any of the ten independent runs. Moreover, its values are always lower than the others, and, in particular, lower than the *star* topology, which is also never selected, despite being a valid topology, suggesting also that the absolute values may represent the validity of a choice.

Equivalent Settings

In the last experiment of this section, the basic representation of a PSO configuration has been modified by adding a fourth topology that, when selected, acts exactly as the *global* topology. The goal was to see if EMOPaT would be able to understand that the two topologies were in fact the same one. Figure 4.5 shows the results in the same format as Figure 4.4. There is no clear correlation between the two "*global*" versions, but it can be seen how, at the end of the evolution, the sum of their selection percentages is close to the value reached by *global* in the previous experiment. It means that, splitting this choice into two distinct values did not affect EMOPaT performance. Nevertheless, a higher number of generations was necessary to reach these results, showing that EMOPaT needs more time to "understand" the correct values of a nominal parameter when several choices are allowed.

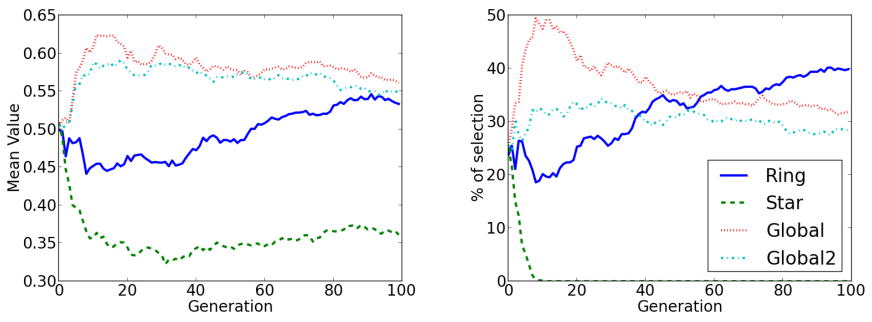


Figure 4.5: Average values assumed by the genes representing the four topologies (including the repeated one) and percentages of selection. The x axis reports the number of NSGA-II generations.

4.1.2 Comparison with SEPaT

The experiments in this section will empirically assess the performance of EMOPaT, by comparing its results with its single-objective version which has been proven to be effective through comparisons with *irace* and with the results of an exhaustive search (see Section 3.1). The comparison has been made on the same seven benchmark functions used in the experiments in Section 3.1.2. Tuning has been performed (i) once for these functions as seven different objectives using EMOPaT, and (ii) seven separate tuning processes with SEPaT, once for each function. More details about these experiments are summarized in Table 4.4.

The goal of these tests is to show that the best-performing parameter sets found by EMOPaT considering a single objective are similar and, consequently, have similar performance to the ones obtained by SEPaT: in other words, EMOPaT works properly if its best solutions on a single objective are indistinguishable from the ones obtained by SEPaT on the same objective.

Ten independent runs for both SEPaT and EMOPaT have been performed. The best configuration of the MH being tuned found in each run for each function was then run 100 times to optimize the corresponding function. The median of these latter experiments was computed for each set of runs to determine the overall best configu-

Table 4.4: Comparison between EMOPaT and SEPaT. Experimental settings.

EMOPaT settings
Population Size = 200, 80 Generations,
Mutation Rate = 0.125, Crossover Rate = 0.9
SEPaT settings
Tuner EA = DE, Population Size = 200, 80 Generations
CR = 0.91, F = 0.52, Mutation = *target-to-best*, Crossover = *Exponential*
Function settings
10-dimensional Sphere, Rotated Cigar, Rotated Rosenbrock, Rotated Ackley,
Rastrigin, Composition Function 1, Composition Function 3 functions
Best fitness in 20000 evaluations averaged over 15 repetitions.

ration for each function. Figure 4.6 shows the ranges of the values found by the two methods for the crossover rate in DE and the inertia factor in PSO: it can be observed that the parameters overlap very frequently.

Table 4.5 compares the best PSO and DE configurations obtained by SEPaT in ten independent runs with the best configurations obtained, for each corresponding function, in ten independent runs of EMOPaT; there is a significant similarity between the parameters obtained by the two methods. For instance, looking at the nominal parameters of both DE and PSO, the choice is the same in all cases except for the PSO topology for Composition Function 3. This is also the only case in which the parameter sets chosen by the two methods are significantly different (one population is three times larger than the other, c_1 is four times bigger and the topology is different): nevertheless, since the results obtained by these two configurations are equivalent (see table 4.6), it can be stated that these sets correspond to two equivalent minima of the meta-fitness landscape.

Table 4.6 shows, for every function, the median fitness obtained by the best-performing of the ten EA configurations obtained by the two methods and by the standard parameters (see Table 3.6), followed by the p-values of Wilcoxon signed-rank test using the Null Hypothesis “There are no differences between the two dis-

Table 4.5: Comparison between EMOPaT, SEPaT and standard parameters. For the two tuning methods, the best-performing parameters obtained over 10 independent runs are shown.

Differential Evolution						
Function	Method	PopSize	CR	F	Mutation	Crossover
Sphere	SEPaT	20	0.506	0.520	*target-to-best*	*exponential*
	EMOPaT	12	0.181	0.718	*target-to-best*	*exponential*
R. Cigar	SEPaT	60	0.955	0.660	*target-to-best*	*binomial*
	EMOPaT	38	0.916	0.699	*target-to-best*	*binomial*
R. Rosenbrock	SEPaT	39	0.993	0.745	*random*	*exponential*
	EMOPaT	47	0.989	0.761	*random*	*exponential*
R. Ackley	SEPaT	85	0.327	0.0	*random*	*exponential*
	EMOPaT	248	0.960	0.0	*random*	*exponential*
Rastrigin	SEPaT	36	0.014	0.359	*random*	*exponential*
	EMOPaT	25	0.049	1.065	*random*	*exponential*
CF 1	SEPaT	18	0.0	1.777	*best*	*exponential*
	EMOPaT	33	0.045	1.070	*best*	*exponential*
CF 3	SEPaT	89	0.794	0.070	*random*	*binomial*
	EMOPaT	98	0.868	0.088	*random*	*binomial*

Particle Swarm Optimization						
Function	Method	PopSize	w	c_1	c_2	Topology
Sphere	SEPaT	88	0.529	1.574	1.057	*global*
	EMOPaT	25	0.774	1.989	0.591	*global*
R. Cigar	SEPaT	67	0.713	0.531	1.130	*ring*
	EMOPaT	41	0.757	1.159	1.097	*ring*
R. Rosenbrock	SEPaT	104	0.597	1.032	1.064	*ring*
	EMOPaT	87	-0.451	-0.092	1.987	*ring*
R. Ackley	SEPaT	113	0.381	0.210	1.722	*ring*
	EMOPaT	115	0.303	-0.006	2.467	*ring*
Rastrigin	SEPaT	13	-0.236	0.090	3.291	*global*
	EMOPaT	7	-0.260	0.021	3.314	*global*
CF 1	SEPaT	92	-0.147	-0.462	2.892	*ring*
	EMOPaT	61	-0.163	-0.376	3.104	*ring*
CF 3	SEPaT	61	0.852	0.347	0.989	*ring*
	EMOPaT	217	0.728	1.217	0.565	*global*

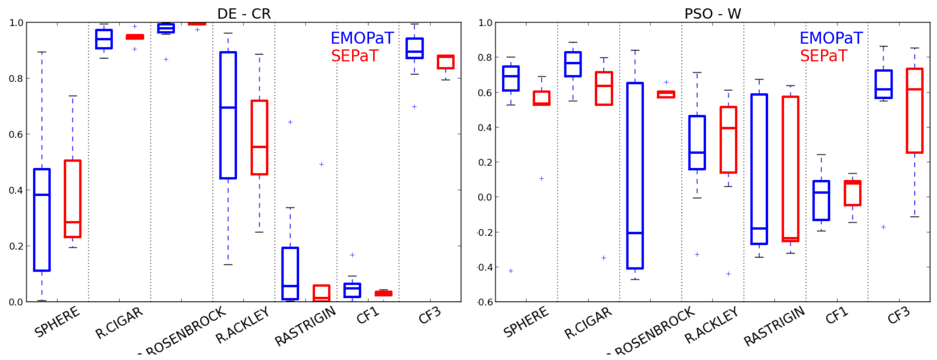


Figure 4.6: Values assumed by crossover rate in DE (left) and inertia factor in PSO (right) after been optimized using EMOPaT (blue boxplots) and SEPaT (red). It is visible that the two methods find similar results.

tributions" comparing EMOPaT against SEPaT and EMOPaT against the standard parameters. While, in most cases, the configurations found by EMOPaT perform better than the standard parameters (last column), there is no statistical evidence that the performances of the configurations found by the two methods differ, except for two cases (Rotated Cigar and Rotated Ackley using DE) for which EMOPaT is slightly better than SEPaT.

These results point out that obtaining the best results for an objective of interest using EMOPaT is generally equivalent to (if not better than) the best results obtained by its single-objective version, with a comparable computational burden. For this reason, the multi-objective strategy can be preferred, since it can provide more information without affecting performance with respect to the single-objective version.

4.1.3 Looking at the Pareto Front

The goal of this experiment is to show that the solutions lying on the Pareto front are valid configurations that represent different trade-offs between the goals considered. To do so, each goal of EMOPaT has been set as the same fitness function whose optimization is limited by a different fitness evaluations budget; similar experiments can be performed evaluating the function under different conditions (e.g. different

Table 4.6: Comparison between the median fitness obtained by the best solutions found by EMOPaT, SEPaT and a standard configuration of the optimization algorithm over seven functions (100 independent runs for each experiment).

EA	Function	EMOPaT	SEPaT	Standard	vs SEPaT	vs Standard
		Fitness			p	
DE	Sphere	0.00	0.00	$7.43E-26$	1.00	$< 1E-20$
	R. Cigar	$7.61E-02$	$7.64E-04$	$1.76E+01$	$4.89E-03$	$5.49E-08$
	R. Rosenbrock	$3.42E-02$	$2.76E-03$	$9.81E+00$	0.41	$< 1E-20$
	R. Ackley	$2.04E+01$	$2.05E+01$	$2.05E+01$	$1.21E-03$	$9.34E-07$
	Rastrigin	0.00	0.00	$2.17E-08$	1.00	$< 1E-20$
	CF 1	$2.04E+02$	$2.05E+02$	$4.00E+02$	0.06	$< 1E-20$
	CF 3	$6.09E+02$	$6.14E+02$	$1.46E+03$	0.67	$< 1E-20$
PSO	Sphere	0.00	0.00	$7.57E-24$	1.00	$< 1E-20$
	R. Cigar	$1.33E+06$	$2.43E+06$	$1.84E+06$	0.05	0.03
	R. Rosenbrock	$9.44E-01$	$1.01E+00$	$9.81E+00$	0.87	$1.04E-17$
	R. Ackley	$2.04E+01$	$2.04E+01$	$2.04E+01$	0.16	0.67
	Rastrigin	$4.75E-06$	$1.02E-04$	$1.09E+01$	0.57	$5.88E-08$
	CF 1	$2.01E+02$	$2.03E+02$	$4.00E+02$	0.99	$< 1E-20$
	CF 3	$9.62E+02$	$9.85E+02$	$1.16E+03$	0.27	$2.7E-04$

problem dimension) or according to different quality indexes. The following set of quality indexes has been considered:

- $\{Q_{X_i}\}$: best results obtained after $\{X_i\}$ fitness evaluations, averaged over N runs;

Two different sets of tests have been performed, in each of which one of the two functions shown in table 4.7 was used. Each one was optimized according to three objectives, namely Q_{1K}, Q_{10K}, Q_{100K}, i.e. the best fitness values reached after 1000, 10000 and 100000 function evaluations. The goal of this experiment is to observe the emergence of configurations that are "fast-converging" or "slow-converging". Table 4.7 summarizes this experiment. Each test was run 10 times.

The hypothesis to be tested is that optimal configurations for intermediate evaluation budgets can be inferred by the results obtained on the objectives that have actually been optimized. These solutions can be generated in two ways:

- infer them as the mean of the two configurations that obtained the best results

Table 4.7: Comparison with different fitness evaluation budgets. Experimental settings.

EMOPaT settings
Population Size = 64, 60 Generations,
Mutation Rate = 0.125, Crossover Rate = 0.9
Function settings
30-dimensional Sphere, Rastrigin (one for each experiment)
Best fitnesses after 1K, 10K, 100K evaluations averaged over 15 repetitions.

for two different objectives between which the new objective lies. This approach presents some limitations: (i) the parameter must follow a clear trend, (ii) a policy for nominal parameters has to be defined if the two configurations used as reference have different settings, (iii) there is no guarantee that all intermediate values of a parameter correspond to valid configurations of the algorithm;

- select them from the Pareto front: after plotting the set of all the population members obtained at the end of ten independent runs on the two objectives of interest, estimate the Pareto front based on those solutions and randomly pick one configuration that lies on it, in an intermediate position between the extremes (see an example on the Rastrigin function in Figure 4.7).

Table 4.8 shows the parameters of the best solutions found for Sphere and Rastrigin for the three objectives and of four intermediate solutions, generated by the two methods. The ones indicated by A lie between Q_{1K} and Q_{10K} and the ones indicated by B between Q_{10K} and Q_{100K}. It can be noticed that parameter sets generated using the Pareto front have different values from the ones inferred as a mean, and from the best configurations. In some cases (such as DE on Sphere) these solutions use a mutation type that differs from all other best configurations. For the nominal parameters of the inferred solutions, when the two best configurations disagree, both options have been considered, distinguishing them by an index (e.g., A_1 and A_2).

Table 4.8: DE and PSO configurations considering as different goals the optimization of the same function with three different evaluation budgets."Best" are the parameter sets that perform best on each objective; "Inferred" refers to the ones obtained as an average of the best solutions; "From Pareto" are extracted from the Pareto front obtained considering objectives pairwise (see Figure 4.7).

Differential Evolution							
	Method	Configuration	PopSize	CR	F	Mutation	Crossover
Rastrigin	Best	Q_{1K}	9	0.214	0.736	*target-to-best*	*binomial*
		Q_{10K}	7	0.053	0.754	*target-to-best*	*exponential*
		Q_{100K}	7	0.039	0.784	*target-to-best*	*exponential*
	Inferred	A_1,A_2	8	0.134	0.745	*target-to-best*	*bin., exp.*
		B	7	0.046	0.769	*target-to-best*	*exponential*
	From Pareto	A	9	0.217	0.763	*target-to-best*	*binomial*
		B	7	0.006	0.769	*target-to-best*	*binomial*
Sphere	Best	Q_{1K}	5	0.022	0.229	*random*	*binomial*
		Q_{10K}	14	0.363	0.508	*random*	*exponential*
		Q_{100K}	30	0.043	0.521	*random*	*exponential*
	Inferred	A_1,A_2	9	0.192	0.368	*random*	*bin., exp.*
		B	22	0.203	0.514	*random*	*exponential*
	From Pareto	A	8	0.023	0.520	*target-to-best*	*exponential*
		B	15	$5E-4$	0.498	*target-to-best*	*binomial*

Particle Swarm Optimization							
	Method	Configuration	PopSize	w	c_1	c_2	Topology
Rastrigin	Best	Q_{1K}	18	0.560	1.195	0.789	*global*
		Q_{10K}	26	0.579	2.492	0.671	*global*
		Q_{100K}	76	-0.251	2.533	0.487	*global*
	Inferred	A	22	0.569	1.844	0.730	*global*
		B	51	0.164	2.513	0.579	*global*
	From Pareto	A	27	0.678	0.949	0.587	*global*
		B	60	0.297	3.132	0.481	*global*
Sphere	Best	Q_{1K}	11	0.603	1.882	1.105	*ring*
		Q_{10K}	14	0.510	1.998	1.483	*ring*
		Q_{100K}	15	0.449	1.725	1.667	*ring*
	Inferred	A	12	0.557	1.940	1.294	*ring*
		B	15	0.480	1.861	1.575	*ring*
	From Pareto	A	14	0.649	1.764	1.082	*ring*
		B	15	0.480	1.681	1.635	*ring*

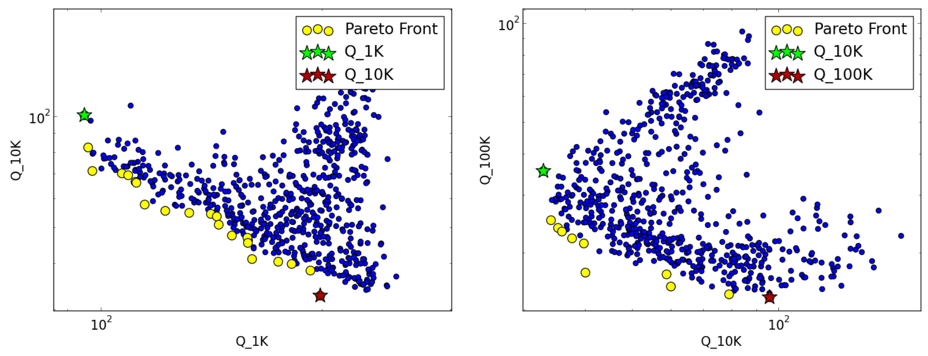


Figure 4.7: Fitness values of all the solutions found in ten independent runs of EMOPaT for the three criteria, plotted pairwise for adjacent values of the budget. The green and red stars represent the best configurations for each objective, yellow circles are candidate optimal solutions for intermediate evaluation budgets.

Figure 4.8 shows the performance of the configurations considered in Table 4.8, averaged over 100 independent runs, for DE and PSO. It can be observed that the intermediate configurations thus generated usually perform as expected. The solid lines represent "Best" configurations; as expected, after 1000 evaluations (highlighted in the plots on the right) Q_{1K} is the best-performing configuration, while Q_{100K} is slower in the beginning but is the best at the end of the evolution. In most cases, the inferred solutions performance can be considered to lie between the two top solutions used as starting points. An interesting result related to the Rastrigin function is reported in the first row of Figure 4.8, considering, for each generation, the best-performing configuration: in the first 1000 evaluations, it is Q_{1K}, then it is surpassed by Inferred A, followed by Q_{10K}, Pareto B, and finally Q_{100K}; this example shows that it is possible to generate new effective solutions starting from the results of EMOPaT. A relevant exception is A_2 in DE Sphere (Figure 4.8, last row) that performs worse than all others: since its only difference with A_1 is the crossover type, this suggests that it is not possible to derive nominal parameters unless one is clearly prevalent.

These results prove that EMOPaT is able to generate an entire set of valid solutions in a single meta-optimization process. In the next section, it will be shown

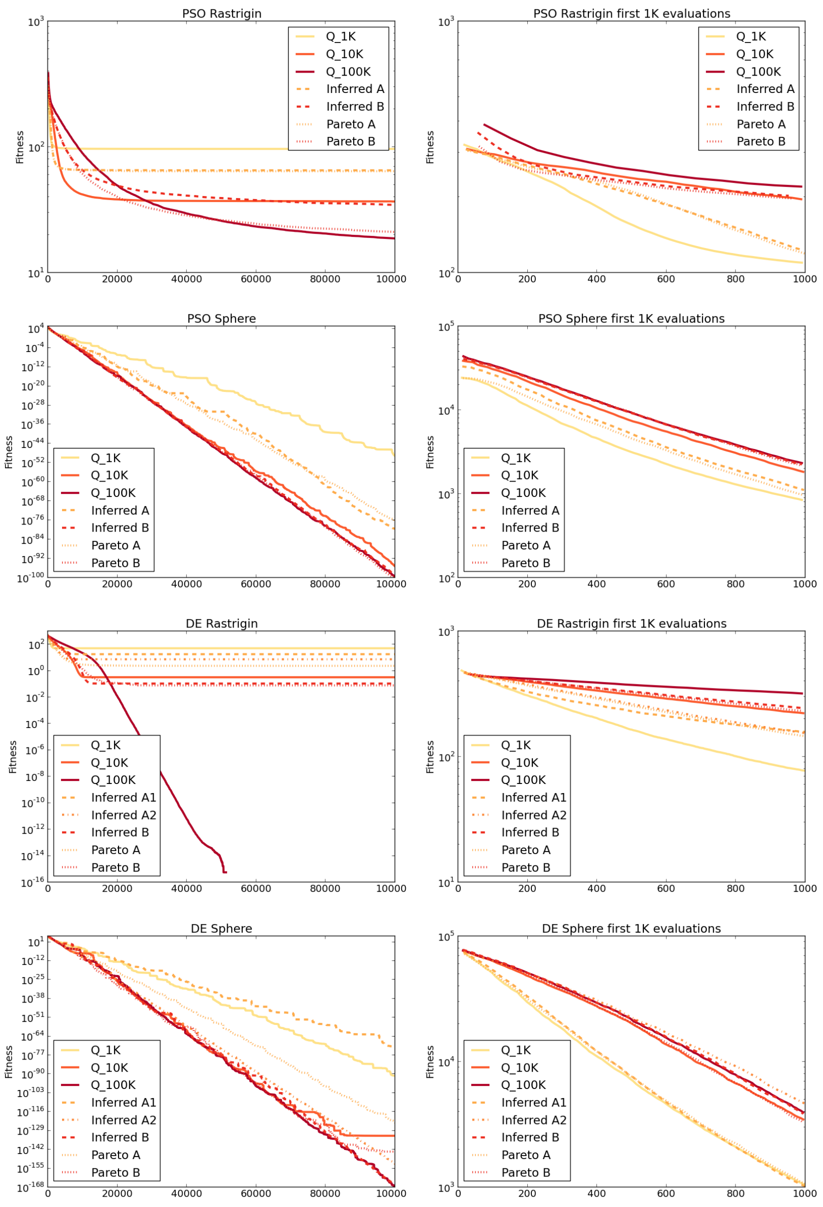


Figure 4.8: Average fitness versus fitness evaluations for the PSO configurations (top two rows) generated for the 30-dimensional Rastrigin (first row) and Sphere (second row) functions. Same plots for DE (last two rows). The plots on the right magnify the first 1000 evaluations to better compare the performance of “fast” versions.

how the analysis of EMOPaT's results can be used to broaden the knowledge of the LL-MH used.

Table 4.9: Tests on EMOPaT. Experimental settings.

EMOPaT settings
Population Size = 60, 60 Generations,
Mutation Rate = 0.125, Crossover Rate = 0.9
Function settings
10- and 30-dimensional Sphere, Rosenbrock, Griewank and Rastrigin functions
Best fitness in $1E6$ evaluations over 15 repetitions and
number of evaluations needed to reach a goal over 15 repetitions
Goals = 0.1 for Sphere and Griewank, 1.0 for Rosenbrock and 10.0 for Rastrigin

4.1.4 Exploiting the Multi-Objective Paradigm

The tests performed in this section take into consideration one function and two quality criteria at one time. EMOPaT has been used to tune DE and PSO over the functions reported in Table 4.9; for each function, the two quality criteria considered are:

Q1 (Precision): best result obtained in $1E6$ fitness evaluations, averaged over N independent runs. To succeed in this criterion, a MH must be able to explore the search space, find a good solution and refine it as much as possible;

Q2 (Speed): number of function evaluations needed to reach a minimum fitness threshold (or goal), averaged over N independent runs. A MH that obtains good results according this criterion must be able to move fast toward a good, even if suboptimal, solution.

Within this experiment, several analyses can be performed by analyzing the results provided by EMOPaT:

1. Comparison between the performance of the automatically-tuned versions of DE and PSO;

2. Detection of a relationship between parameters and quality criteria;

3. Analysis of the range of the parameters of DE and PSO found by EMOPaT;

4. Comparison between the parameters obtained by EMOPaT and the parameters commonly used in the literature (or obtained using other techniques).

The first thing that can be noticed is that the Sphere function, which is a simple and unimodal function, and to a lesser extent Rosenbrock, obtained fronts usually composed by less than three, almost indistinguishable, solutions: this means that, as could have been expected, in these cases a greedy algorithm is the best choice because there is simply no trade-off between exploration and exploitation of the search space, and a fast algorithm produces good results using a limited number of function evaluations. In other words, the two objectives are not in conflict, therefore the Pareto front is reduced to a single solution.

In the following paragraphs, the four points just described will be analyzed one at a time and some examples will be provided to show the potentiality of EMOPaT.

Comparison between DE and PSO

A straightforward analysis that can be made when running EMOPaT with two algorithms (in this case DE and PSO) is a direct comparison between the Pareto fronts obtained by optimizing their parameters. Figure 4.9 compares the fronts obtained on the Rastrigin and Griewank functions (10 and 30 dimensions) for DE (in dark red) and PSO (in light blue) in the 10 runs. It can be seen how, in both functions, DE is better than PSO according to both objectives, since PSO generates solutions that are almost always dominated by those obtained using DE. What is important to notice is that both MHs have been tuned in the same way, therefore this comparison is not subject to biases introduced by the developer.

Parameters and quality criteria

In order to show how parameters affect the quality criteria, i.e. their positions on the Pareto fronts, two subsets of solutions have been selected:

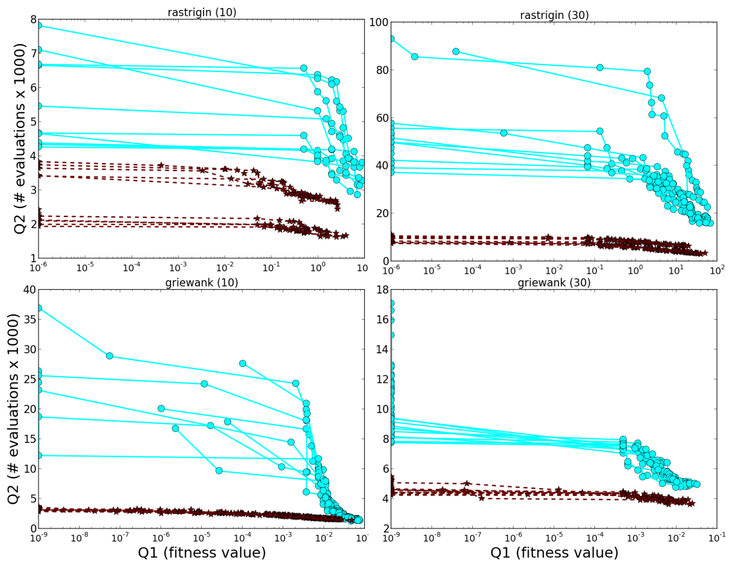


Figure 4.9: Pareto fronts discovered by EMOPaT (dotted dark red lines for DE, solid light blue lines for PSO) for Rastrigin (top) and Griewank (bottom) functions in ten (left) and thirty (right) dimensions. It can be seen that solutions found by DE are usually better than the ones obtained by PSO according to both metrics. Results of Q1 below a low threshold have been clipped for visualization purposes.

- *S1* which comprises the 10% of solutions with best results according to *Q1*;
- *S2* in the same way, but considering *Q2*.

A significant difference in the values of a parameter between the two subgroups is a reliable suggestion that such a parameter plays an important role in the algorithm. Figure 4.10 compares some properties of *S1* and *S2* for DE, on the Rastrigin and Griewank functions and shows that some parameters assume very different ranges within the two subgroups.

An interesting outcome is that the Sphere function, compared to the other two (multimodal) ones, never selects *random* mutation, in favor of *target-to-best* and *best*. This confirms the hypothesis that *random* mutation has a less greedy behavior than

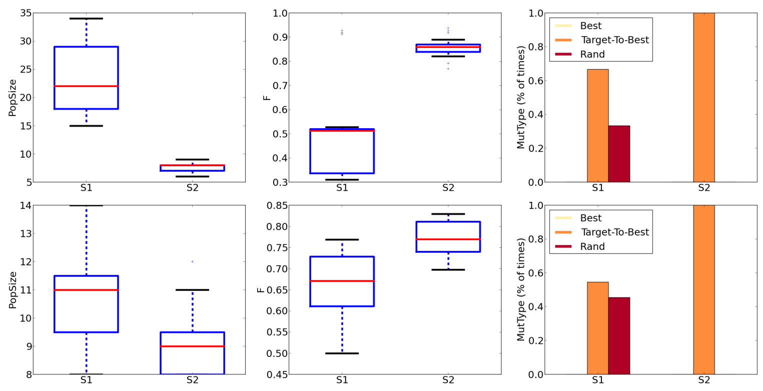


Figure 4.10: In all boxplots and bar plots, a comparison is made between the subsets *S1* and *S2*. Each boxplot represents the distribution of the parameter represented on the y axis in the corresponding subgroup. The bar plots on the right describe the distribution of the selected mutation types for the 30-dimensional Rastrigin (top row) and Griewank (bottom) functions; from left to right Population Size, F (scale factor) and mutation type. The distribution of these three parameters is different in the two subgroups, indicating that they significantly affect DE's behavior.

the other two. Therefore, EMOPaT's results suggest that it seems possible to order the three kinds of mutations considered based on increasing greediness (or decreasing precision): the greediest (therefore, the least precise) is *best*, then *target-to-best* and finally *random* mutation.

The most evident result that recurs in all experiments, for both PSO and DE, is the direct relationship between population size and the location of a MH instance on the Pareto front: an algorithm with more elements wastes fitness evaluations in sub-optimal zones, slowing down convergence, but small populations are often trapped in local minima (see also [82]), making it impossible to consistently reach good results. Figure 4.11 shows, for the Griewank and Rastrigin functions, that *Q2* and population size are directly correlated for the two MHs. This result confirms the ones obtained with DE by Mallipeddi et al [96]. Another interesting result is that DE needs a smaller population than PSO while, for both, it seems that the population size needs to be

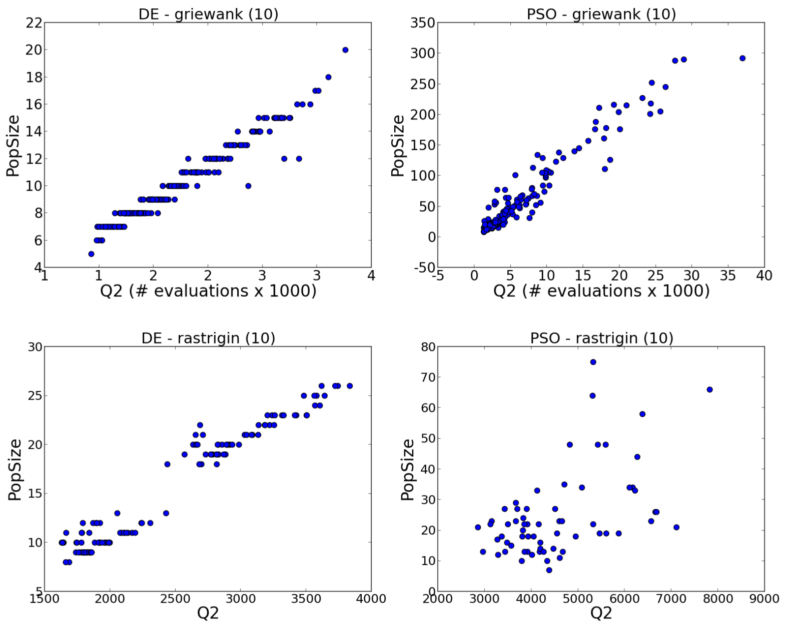


Figure 4.11: Population Size versus *Q2* for DE and PSO. The relationship between the parameter and the result is clear to the naked eye, and it is confirmed by computing the Pearson coefficient, whose value is > 0.96 in all cases except Rastrigin 10 ($r = 0.52$). From this comparison it is also noticeable that PSO needs a larger population (and consequently more fitness evaluations) to reach the same fitness as DE.

quite small anyway; classic rules of thumb used (e.g. population size equal to 10 times the problem dimension) seem to be contradicted, in favor of more rigorous studies [44]. A similar observation can be made for F, particularly for the Griewank function: in this case the relationship is inverse, as bigger values of F lead to a faster achievement of the fitness goal, worsening *Q1* (again, see Figure 4.10).

Range of selected parameters

As previously said, PSO and DE parameter values are allowed to vary within a wide range to see if EMOPaT is able to find unexpected behaviors. This is certainly the

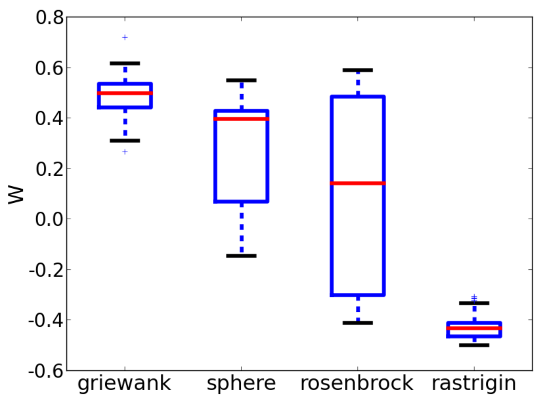


Figure 4.12: Values of inertia factor *w* across the four benchmark functions in 10 dimensions. Negative values are often selected.

case of PSO's inertia factor *w*. In all the solutions in the Pareto fronts of the Rastrigin function and for several sets obtained for the Sphere and Rosenbrock functions, this value is negative, as shown in Figure 4.12. This has been investigated very few times in the past, but hints suggesting this possibility have already been given by automatic tuning methods [134] and systematic searches [99]. It seems that the tuning of this parameter is more influenced by the fitness landscape and less by the quality criterion considered.

Comparison with other parameters

The last analysis regards the comparison of the results obtained by EMOPaT with other reference implementations of PSO and DE. For both MHs, some parameter sets available in the literature have been selected, either manually or automatically tuned by the authors. The goal of this procedure is not to prove the superiority of the solutions found by EMOPaT, since different conditions (such as functions, quality criteria, ...) may need different parameter sets; the main goal is to show that the parameters obtained, and consequently the conclusions provided, are reliable and yield results that are comparable to the state-of-the-art.

To do so, five runs with each reference parameter set (see Table 4.10) have been run. Each of these runs consist of $N = 15$ independent executions of the function under consideration, exactly as it would have happened if these sets of parameters had been generated by EMOPaT during its evolution.

Table 4.10: DE and PSO versions used as reference. The first column shows the name that is used in the text to indicate that particular version of the MH.

Name	Tuning	Ref.	Parameters
PSO_{M1}	Manual	[72]	$Pop.Size = 60$, $c_1 = c_2 = 1.496$, $w = 0.730$, Global Topology
PSO_{M2}	Manual	[72]	$Pop.Size = 60$, $c_1 = c_2 = 1.496$, $w = 0.730$, Ring Topology
DE_{M1}	Manual	[158]	$Pop.Size = 50$, $CR = 0.9$, $F = 0.5$, Random Mutation, Exp. Crossover
DE_{M2}	Manual	[158]	$Pop.Size = 50$, $CR = 0.9$, $F = 0.5$, Random Mutation, Bin. Crossover
PSO_{A1}	Auto	[134]	$Pop.Size = 134$, $c_1 = 1.8903$, $c_2 = 2.122$, $w = -0.1618$, Global Topology
PSO_{A2}	Auto	[117]	$Pop.Size = 125$, $c_1 = 1.862$, $c_2 = 1.881$, $w = 0.494$, Global Topology
PSO_{A3}	Auto	[173]	$Pop.Size = 34$, $c_1 = 1.525$, $c_2 = 1.881$, $w = 0.443$, Global Topology
DE_{A1}	Auto	[134]	$Pop.Size = 19$, $CR = 0.122$, $F = 0.4983$, Random Mutation, Bin. Crossover
DE_{A2}	Auto	[117]	$Pop.Size = 48$, $CR = 0.879$, $F = 0.520$, Random Mutation, Exp. Crossover

Most settings were unable to even reach the fitness goal, especially the manually-tuned ones. Only in a few cases, some of the references obtained results comparable to the versions generated by EMOPaT (see Figure 4.13 for some examples):

- Sphere: for 10 dimensions, DE_{A1} and PSO_{A3} provide good *Q1* with a higher *Q2*; DE_{A1} also prevails in 30 dimensions while performance of PSO_{A3} degrade;

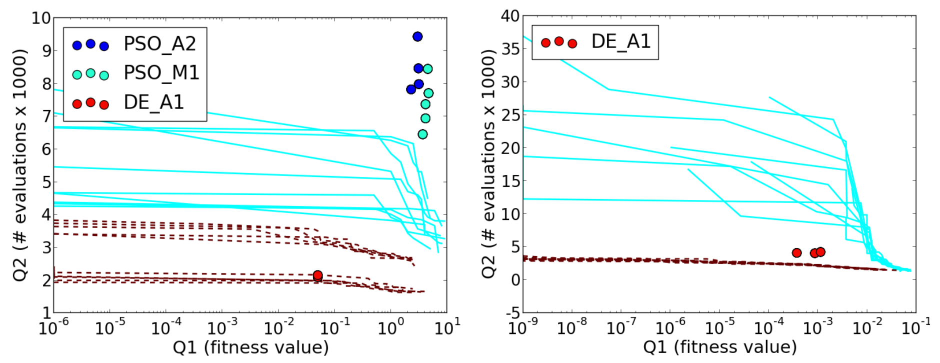


Figure 4.13: The reference versions of DE and PSO that are able to obtain good results on the ten-dimensional Rastrigin function (left) and on thirty-dimensional Griewank (right). The good results of DE_{A1} on Rastrigin are not totally unexpected, because its parameters are very similar to the ones obtained by EMOPaT.

- Rosenbrock: no comparable methods;
- Griewank: DE_{A1} is able to obtain very precise results using a higher number of function evaluations to reach the fitness goal, both in 10 and 30 dimensions;
- Rastrigin: performance of DE_{A1} are comparable to the results obtained by EMOPaT in 10 and 30 dimensions. Two versions of PSO (PSO_{A2} and PSO_{M1}) are slightly worse than those lying on the Pareto front of PSO in the 10-dimensional version, but their performance degrades when increasing the problem dimension to 30.

4.2 Discussion

In this Chapter, EMOPaT has been presented and tested under different conditions, to show its ability to:

1. provide good parameter sets that are able to deal with conflicting goals;

2. provide more knowledge about a MH, in particular about the relationship between a MH and its parameters, and how they affect the performance of the algorithm. In turn, this result may be used for many different purposes, like a fair and unbiased comparison between different (versions of) algorithms. Additional experiments proving this point can be found in [164, 166].

At the moment, the main drawback of EMOPaT is that the analysis of the results is still a "manual" piece of work. A possible way to automatically extract, generalize and infer parameters is the one presented as automated *innovization* [31], in which the *innovization* process is completed by other analyses such as clustering to automatically identify the parameters that most influence the results. Nevertheless, a "manual" work by an experienced developer could be sufficient to exploit most of the information provided by EMOPaT.

More details about the experiments presented in this Chapter, as well as additional ones, can be found in:

- R. Ugolotti and S. Cagnoni. "Analysis of Evolutionary Algorithms using multi-objective parameter tuning". In: Genetic and Evolutionary Computation Conference (GECCO'14), pages 1343-1350. ACM, 2014
- R. Ugolotti and S. Cagnoni. "Design of EAs for Continuous Optimization aided by Multi-Objective Parameter Tuning". Submitted to Evolutionary Computation

Chapter 5

Model-based Object Recognition

Science isn't about authority or white coats;
it's about following a method.

– Ben Goldacre

This Chapter introduces the general framework for template-based object recognition in images and videos using bio-inspired techniques. Then, several examples of applications of this method developed during the last three years will be presented. In the last two cases, the automatic parameter tuning methods presented in the previous Chapters have been used to successfully improve the results.

5.1 General Framework

In section 1.2, some possible general methodologies have been presented that can be used to automatically find an object inside an image on track it in a video. The model-based approach uses a template describing the possible appearances of the object and is then used to recognize/localize it within the current image or frame. To apply this procedure to a particular object, or class of objects, one needs to define [174]:

1. A template of the object. This template must take into considerations the most

relevant intrinsic features of the object (usually based on the observation of a significant subset of object instances) that are of interest and ignore the others; for instance, if the color is a key descriptor of the object, the template must contain this information, while it should be ignored in the opposite case;

2. A set of actions that the template can be subject to (translations, rotations, deformations, . . .) in order to match the object appearance, and the limits of the ranges of these actions, based on the knowledge of the physical laws which regulate the object and the image acquisition process;

3. A way to generate candidate solutions starting from the template;

4. A similarity (or distance) function between the candidate solution and the object to be observed that reaches its maximum (minimum) when the image representation of the template is perfectly superimposed to the object as it appears in the image. The same rule considered in the first point must be used. This function must give more importance to the factors that are of most interest within the application: in some cases a precise localization is more important, in others the similarity (difference) between the template and the object.

Table 5.1 summarizes how the four points listed above have been dealt with in the four applications that will be presented in the following sections.

The only point these applications have in common is that the generation of possible solutions is performed using bio-inspired optimization since, as shown in the introduction, this can be considered a global optimization problem, and usually a highly multimodal one. Let us use the problem of hippocampus localization in histological images that will be presented in Section 5.4 as an example: this is a 14- to 16-dimensional problem but, even considering only the first two (which represent the position of the template within the image), it already presents several local optima, as shown in Figure 5.1.

Automatic parameter tuning surely helps to improve the performance of this procedure, because it can be applied only once on a small subset of possible instances of

Table 5.1: Summary of the four object recognition applications. For each application, it shows how the template is defined, the possible transformations it can undergo, how candidate solutions are generated, and the criterion followed by the similarity function.

Application	Sect.	Template	Transformations	Generation	Similarity
Road Signs Detection	5.2	Three sets of points	Rigid transformation	Bio-inspired Optimization	Color-based
Human Body Pose Estimation	5.3	Deformable template (skeleton + mesh + dimensions)	Rotations of the limbs, size and position of the model		Distance cylinder-points
Hippocampus Localization	5.4	ASM-like deformable template	Model deformation		Energy-based
Point Cloud Localization	5.5	Point cloud	Rigid transformation		Point cloud distance

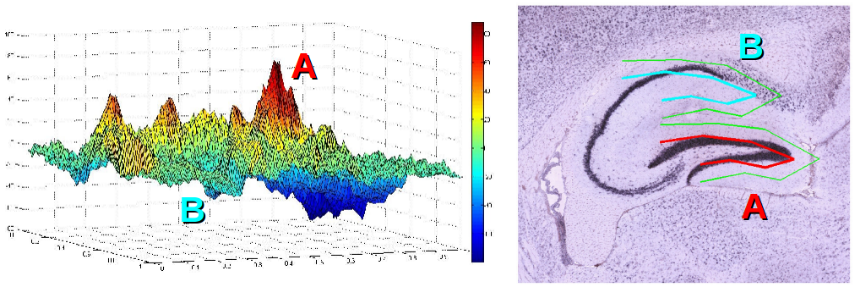


Figure 5.1: Variability of the target function versus the initial position of the template. The model labeled as A represents a good localization of the lower part of the hippocampus, and is associated with high values. In opposition, model B is badly located, so its value is low. The fitness landscape (left) is highly multimodal.

the object to be tracked and can find parameters that will be performant over different unseen instances.

The remaining of this Chapter describes the four different implementations of this approach, dealing with four very different tasks, which prove the wide applicability of this simple paradigm. In the last two applications, it will also be shown how an automatic parameter tuning procedure is helpful in improving the performance of the system.

5.2 Road Signs Detection

Automatic road sign detection and classification is a task that can help drivers and increase road safety. For this reason, this problem has been frequently tackled [109]. Solutions to this problem usually include two different stages: the presence of a sign is first detected in the image, then it is classified to precisely recognize its meaning and possibly activate some driving system control. In the detection phase, the features used most frequently to recognize a sign are shape and color, but a combination of the two is usually preferred. The techniques most frequently used for classification are artificial neural networks and support vector machines.

The work presented in this section has been originally developed by Mussi et al [113] and then expanded in [172].

5.2.1 Model and Similarity Function

Four models have been developed (see Figure 5.2), each of which aims at recognizing a particular class of signs (priority, warning, prohibitory and mandatory). Each model is composed of three sets of reference points, usually placed just near the color

More details about the experiments presented in this section can be found in

- R. Ugolotti, Y.S.G. Nashed, and S. Cagnoni. “Real-time GPU based Road Sign Detection and Classification”. In: Parallel Problem Solving From Nature - PPSN XII, volume 7491 of Lecture Notes in Computer Science, pages 153-162. Springer, 2012

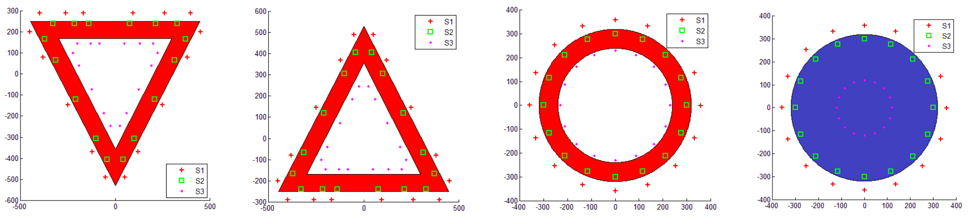


Figure 5.2: The models used to recognize priority, warning, prohibitory and mandatory signs. The first three models consist of three sets of points. The first (S_1) lies just outside the sign; the second (S_2) on the red band, while the third (S_3) inside the sign. In the last case, S_1 behaves in the same way, S_2 lies on the border of the sign and S_3 in the innermost part.

discontinuities. During the localization phase, the template is roto-translated and projected on the image acquired by a calibrated on-board camera. For every set of points located by the template points after the transformation, three color histograms in the HSV color space are computed.

For the first three models, the distance function is:

$$f = \frac{k_0(1 - B(h_1, h_2)) + k_1(1 - B(h_2, h_3)) + k_2 B(h_1, h_r)}{k_0 + k_1 + k_2}$$

where h_i is the histogram of set S_i, and h_r is a reference histogram centered on red; k_0, k_1 and $k_2 \in \Re^+$ are used to weigh the elements of the equation; $B(x, y)$ is the Bhattacharyya coefficient [69], which estimates the overlap between two statistical samples. If f falls below a pre-defined threshold a sign is detected, which happens when:

- the histogram of the set outside the sign is different from the histogram of the set located on the red band;
- the histogram of the points on the red band is as different as possible from the one computed on the inner area of the sign;
- the histogram of the points on the red band is similar to the reference histogram defined for the red hue (a Gaussian centered on pure red).

The distance function for the mandatory (blue) signs is slightly different:

$$f = \frac{\bar{k}_0(1-B(h_1,h_2)) + \bar{k}_1(B(h_3,h'_r)) + \bar{k}_2 B(h_2,h''_r) + \bar{k}_3(1-B(h_2,h_r))}{\bar{k}_0+\bar{k}_1+\bar{k}_2+\bar{k}_3}$$

where h'_r is a reference histogram with peaks corresponding to blue and white, h''_r a reference histogram centered on blue, and $\bar{k}_i$, $i = 1,\ldots,4$ are positive weights. To be recognized, a sign must have a blue border and a white and blue inner part.

After a sign has been detected, the image region located by the model is then rectified via a inverse perspective transform in order to obtain a frontal view to simplify a neural network-based classification.

5.2.2 Experimental Results

The entire process (also including the neural network-based classification of the sign) has been implemented on GPU. PSO and DE implementations rely on the library described in the Appendix.

Table 5.2: Results of the detection phase (min-max) for the four categories of signs (detections): worst and best result in 10 independent runs.

		Parma Sequence			Turin Sequence		
		Total	False Positives	Detections	Total	False Positives	Detections
Warning	DE	51	0-1	27-31	53	0-2	39-43
	PSO	51	0-0	27-30	53	0-1	35-40
Prohibitory	DE	44	2-6	26-30	47	2-4	39-42
	PSO	44	0-1	22-27	47	0-1	39-40
Priority	DE	30	5-11	18-22	15	2-4	13-15
	PSO	30	0-2	15-18	15	0-1	7-12
Mandatory	DE	62	2-4	40-41	39	0-1	27-29
	PSO	62	0-1	35-39	39	0-1	24-27

The benchmark used to evaluate the results in a real environment is composed of two sequences [101]. The first one, which includes 10000 frames at a resolution of 750×480 pixels, was acquired at 7.5 fps in Parma on a sunny day. The sequence contains images featuring all possible light orientations. The second sequence is about

5000 frames long and was acquired at 7.5 fps in Turin on a cloudy day. Images in this sequence feature more constant lighting but lower contrast. Table 5.2 reports the best and worst detection results obtained over 10 runs on each sequence using DE and PSO. These results are similar to the ones reported in [101], but thanks to the GPU-based implementation, they can been obtained in a shorter time.

5.3 Human Body Pose Estimation

Markerless body pose estimation is an important task in computer vision. The possible applications of this task are countless: among the most common there are human-computer interaction, gaming and medical assistance.

The research in this field has recently made significant progress, thanks to the wide availability of low-cost systems for computing depth maps, such as the Kinect by Microsoft. The availability of depth maps that can be acquired in form of point clouds can limit the effects of some problems that hamper human pose estimation in images, by simplifying tasks like background subtraction, and totally solving others, like depth estimation.

5.3.1 The Model

The three-dimensional model used to represent a human pose can be separated into three different, albeit related, components [165].

The skeleton

The skeleton is defined as a tree whose nodes are 4×4 transformations. Each node represents the relative position and orientation of a joint with respect to its parent

More details about the experiments presented in this section can be found in

- R. Ugolotti, and S. Cagnoni: "Differential Evolution based Human Body Pose Estimation from Point Clouds". In: Genetic and Evolutionary Computation Conference (GECCO'13), pages 1389-1396. ACM, 2013

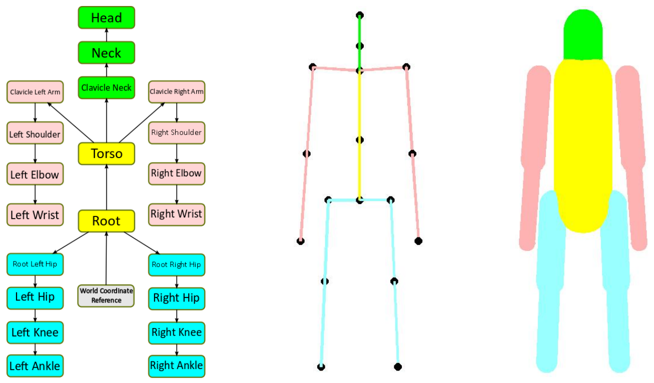


Figure 5.3: On the left, the kinematic chains used to describe the human model. On the centre, the skeleton of the model. This is the base position of the model in the first frame. On the right, the base mesh generated by the skeleton.

node. A joint (root) corresponding to the pelvis is the top of the hierarchy. Five different kinematic chains are linked to it to describe the entire body. Figure 5.3a describes the kinematic model used for the skeleton, which is shown in Figure 5.3b. The distances between two joints are based on the size of a standard model and are tuned according to the evolution of the *Dimensions* parameter vector (see next paragraph). Since the anatomical limits of the human body do not allow some rotations, the degrees of freedom of the whole skeleton are only 29.

Dimensions

Dimensions is a vector of 7 parameters that define the actual size of the mesh. The reference model is defined as an average-height man, and these scaling parameters affect the distances between the joints and the thickness of the mesh built around them. These parameters are used as multipliers of the standard mesh, i.e., when they are equal to 1, the reference model is used as is; when they are larger than 1 the model is taller/fatter than the standard and when they are smaller than 1 the model is shorter/thinner.

Mesh

The actual mesh (Figure 5.3c) is composed of 11 cylinder-spheres [10], whose positions depend on the location of the skeleton's joints and whose widths depend on the *Dimensions* parameters. Two of them are used to describe the torso, one represents the head, and the remaining eight stand for the arms and the legs.

In conclusion, the total number of parameters subjected to DE optimization is 29 angles + 7 scale factors + 6 values that represent the position and orientation of the model with respect to the world reference system, for a total of 42 parameters. The model is intrinsically hierarchical, and this feature is also exploited during the optimization. In fact, the optimization is not performed all in one step, but is split into four steps, in each of which only some parameters of the model are optimized. In this way, one high-dimensional problem can be seen as four lower-dimensional problems, at the cost of performing four different and consecutive optimization processes. After one optimization step ends, its resulting parameters are frozen and the next step considers them as constants. This choice has been made after some tests made clear that tackling the pose estimation problem in a single step led to large errors.

5.3.2 The distance function

The fitness of a candidate solution is proportional to the sum of the distances of all points in the input cloud from the mesh. For each point in the cloud, the distance from each cylinder that composes the mesh is calculated using a variant of the method presented in [10]. This method has been chosen due to the very light computation needed to calculate the distance between a solid and a point. The shortest distance (d_i) from each point i to the closest part of the mesh undergoes this simple equation:

$$d'_i = \begin{cases} {d_i}^{\alpha} & \text{if } d_i \leq \beta \\ \beta^{\alpha} & \text{otherwise} \end{cases}$$

The two parameters α and β affect the precision of the localization. A value of $\alpha > 1$ can be used to help the model ignore small details, but be more focused on reaching all parts of the body (particularly hands and feet) while a value of $\alpha < 1$

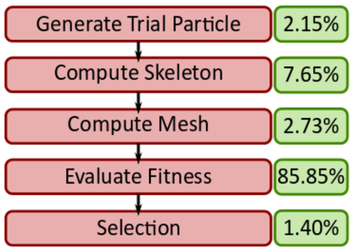


Figure 5.4: The repetition of these five steps compose the body pose estimation process. On the right the percentage of time spent on every function is reported. The total does not add up to 100% because initialization and result retrieval are not included.

induce the model to refine good solutions. β can be seen as the distance beyond which a point does not affect the pose estimate any more. In this way, the behavior of the model becomes more robust with respect to outliers. The final fitness is then calculated as the sum of all d's terms.

5.3.3 Implementation details

Like all methods described in this Chapter, this application has been implemented on GPU. In this way, the huge amount of computation needed can be performed in a fast way and reach real-time performance (average 43.9 ± 3.4 ms for a frame). The final architecture is composed of 5 different functions (see Figure 5.4). The first and the last implement the DE algorithm, while the remaining three are in charge of pose estimation and fitness evaluation:

- “Compute Skeleton” reads all the DE elements to be evaluated in parallel and uses them to generate the location of the skeleton joints. Despite the hierarchical structure of the model, all joints can be computed in parallel in order to save time. A joint that depends on other joint locations, must go back to the root of the hierarchy and compute the chain;

- “Compute Mesh” generates the positions of the cylinders that compose the mesh using the joint locations and size modifiers;

- "Evaluate Fitness" computes, for each point, the distance from the closest part of the mesh and adds them up to obtain the final fitness. This is by far the most computationally expensive part of the algorithm.

5.3.4 Experimental Results

This algorithm has been tested on a publicly available dataset [45]. It is composed of 28 sequences collected using a Time-of-Flight camera, with a resolution of 176×144 pixels. During the acquisition, the subject was wearing markers, whose location can be used as ground truth to assess the precision of the pose estimation. This dataset has also been used in [6, 45, 161, 190], which allows a comparison with the results reported therein to be made.

Every sequence was processed 10 times; the average results (compared to the other four approaches) are presented in Figure 5.5. The sequences of this dataset are approximately sorted by difficulty. While in the first ones the subject performs only small movements, in the last ones the movements are more complicated (including a 360° rotation and a tennis serve). Our method is one of the worst performing on the simpler sequences (probably due to our mesh definition, since a cylinder is not a perfect approximation of a limb) while, when processing the most complicated ones, results are comparable to the state-of-the art, showing that our method has good generalization properties and good tracking results.

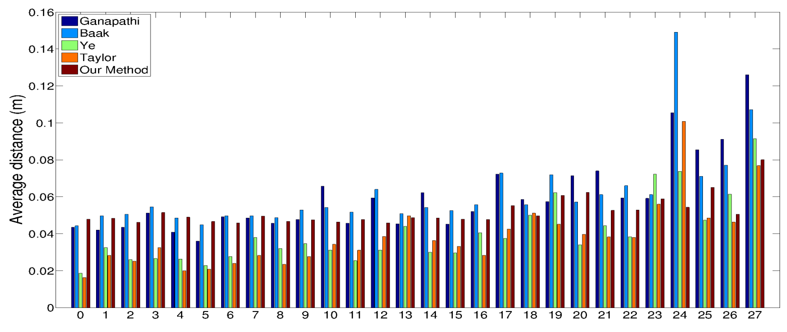


Figure 5.5: Comparison with the state-of-the-art. Results represent the average distance over all markers on all frames of the sequence over 10 independent runs.

5.4 Hippocampus Localization

The hippocampus is a structure located in the mammalian brain that has long been known for its crucial role in learning and memory processes [123], as well as an early biomarker for Alzheimer disease and epilepsy. Thus, its automatic, robust and fast localization is of great interest for the scientific community. From an anatomical point of view, the hippocampus (see Figure 5.6) is located within the medial temporal lobe and it is composed by the Dentate Gyrus (DG) and Ammon's Horn (CA), which is further composed by three different regions (CA1, CA2, and CA3). In turn, within these regions, the zones that are visually easiest to locate are the pyramidal (*SP*) and granule (*SG*) cell layers, which belong to the CA and DG regions, respectively.

The goal of the project [170] within which the application presented here was developed was to automatically locate and segment the hippocampus in mouse brain histological images, in order to extract textural information and find genes with visual patterns (and consequently, chemical behavior) similar to other genes with known

More details about the experiments presented in this section can be found in the following publications.

- P. Mesejo, R. Ugolotti, F. Di Cunto, M. Giacobini, and S. Cagnoni. "Automatic Hippocampus Localization in Histological Images using Differential Evolution-Based Deformable Models". In: Pattern Recognition Letters, Volume 34(3), pages 299-307, 2012
- R. Ugolotti, P. Mesejo, S. Zongaro, B. Bardoni, G. Berto, F. Bianchi, I. Molineris, M. Giacobini, S. Cagnoni, and F. Di Cunto: "Visual Search of Neuropil-Enriched RNAs from Brain In Situ Hybridization Data through the Image Analysis Pipeline Hippo-ATESC" In: PloS-One, Vol. 8, 2013
- P. Mesejo, R. Ugolotti, F. Di Cunto, S. Cagnoni, and M. Giacobini. "Automatic Segmentation of Hippocampus in Histological Images of Mouse Brains using Deformable Models and Ensemble Classifiers". In: IEEE International Symposium on Computer-Based Medical System (CBMS'12), 2012
- R. Ugolotti, P. Mesejo, Y.S.G. Nashed, and S. Cagnoni: "GPU-Based Automatic Configuration of Differential Evolution: A Case Study". In: Progress in Artificial Intelligence, Volume 8154 of Lecture Notes in Computer Science. Springer, 2013

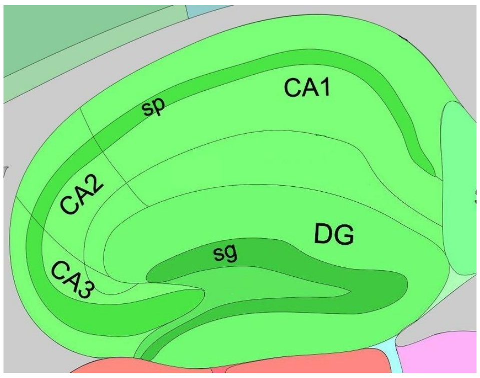


Figure 5.6: Regions of the hippocampus.

properties used as reference. A correct localization of the hippocampus is therefore a non-trivial but essential part of this pipeline, summarized in Figure 5.7.

The images used in this study have been extracted from the Allen Brain Atlas (ABA), a publicly available image database [84] which contains a genome-scale set of histological images (cellular-resolution gene-expression profiles) obtained by In Situ Hybridization of serial sections of mouse brains. Figure 5.8 reports some examples taken from this source, showing the significant variability of the appearance of hippocampi, in term of size, shape, hue, lighting and considering the possible artefacts due to image acquisition like tears, scraps and bubbles.

5.4.1 Template and Fitness Function

The template used in this application [168] is based on Active Shape Models. It consists of a medial shape-based representation of the hippocampus in polar coordinates, and has the objective of creating simple models that can be managed easily and efficiently. Two parametric models representing *SP* and *SG* are moved and deformed by DE according to an energy-based similarity function between the template and the hippocampus image.

Each model comprises two sets of points (Figure 5.9a): the goal is to superimpose the first one (Inner Set, $\vec{I}$) to the part of the hippocampus to be located, while placing

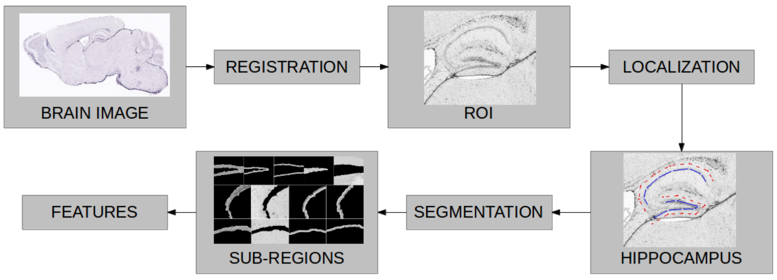


Figure 5.7: The image of the brain is downloaded from the ABA, then an atlas-based registration is used to roughly locate a ROI containing the hippocampus, which is properly localized using the technique described in this section. This localization is used as a starting point for a segmentation step to localize 13 sub-regions of interests, on which a texture analysis is performed to obtain features which represent gene's activation.

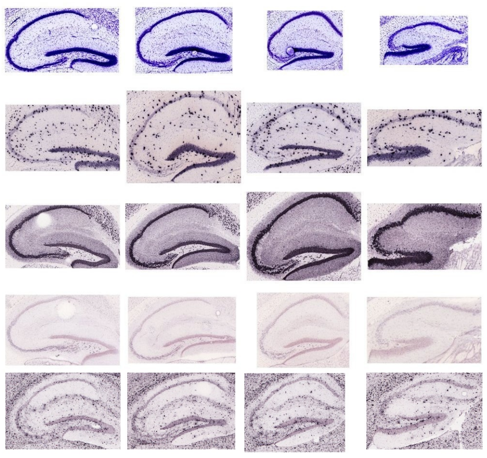
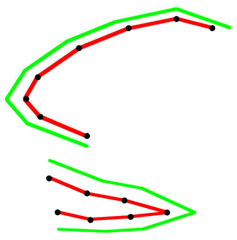
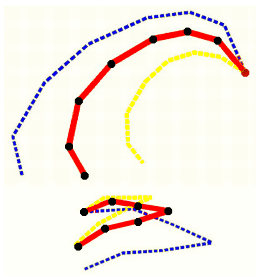
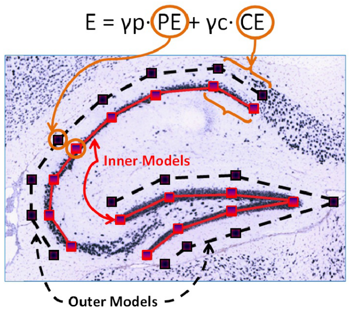


Figure 5.8: Examples of hippocampus images taken from the ABA. From top to bottom, 5 different genes; from right to left, 4 different locations within the brain.


(a)


(b)



(c)

Figure 5.9: (a) the inner and outer models for *SP* and *SG*; (b) the deformations allowed by the inner model (the deformation of the outer model is obtained by a rigid shift of the deformed inner model); (c) the models superimposed to a hippocampus image, showing how the external energy is computed [105].

the other one (Outer Set, $\vec{O}$), obtained by rigidly shifting the first one, immediately outside it. The template is subject to external forces (generated by the attraction of the image features onto the model) and internal forces (generated by the model deformation). The target function H to be maximized (see [106] for a more detailed description) has three components: the external energy EE, the internal energy IE, and the contraction factor C.

$$H = EE - (IE + C)$$

In turn, EE can be divided in two components: PE, that depends on the control points of the model (denoted by black dots in Figure 5.9b) and CE, that is computed considering the points which belong to the segments connecting them:

$$\begin{aligned}
EE &= \gamma_P \cdot PE + \gamma_C \cdot CE \\
PE &= \sum_{i=1}^{n} [T(N_3(I_i)) - T(N_3(O_i))] \\
CE &= \sum_{i=2}^{n} \sum_{j=1}^{p} T(I_{i-1} + \frac{j}{p+1}(I_i - I_{i-1})) - \sum_{i=2}^{n} \sum_{j=1}^{p} T(O_{i-1} + \frac{j}{p+1}(O_i - O_{i-1}))
\end{aligned}$$

where I_i and O_i represent the points of the two sets, γ_P and γ_C are positive values that weigh the two components, $N_3(P)$ is a 3×3 neighborhood centered in P, $T(P)$ is the intensity of P if P is a point, or the average intensity if P is a neighborhood, and p is the number of points sampled in each segment. In short, the main goal is to superimpose $\vec{I}$ to a dark part of the image and $\vec{O}$ to a bright part, right next to $\vec{I}$.

The internal energy IE is computed as:

$$IE = \xi_\rho \cdot \sqrt{\sum_{i=2}^{n} (\rho_i - \rho_{mi})^2} + \xi_\vartheta \cdot \sqrt{\sum_{i=2}^{n} (\vartheta_i - \vartheta_{mi})^2}$$

where ξ_ρ and ξ_ϑ are two positive weights that regulate the deformability of the model. (ρ_i, θ_i) represents the relative position of a point of the model with respect to the previous one in polar coordinates, while (ρ_{mi}, θ_{mi}) represents a point of a reference model derived empirically from a set of "training" images which represents the "expected" shape of the template.

Finally, the contraction factor C also regulates the template's deformability to avoid infeasible situations that are not allowed by the nature of the hippocampus and is defined as follows:

$$C = \xi_c \cdot \|I_n - I_1\|$$

If $\xi_c < 0$ the two extremes of the model repel each other, if $\xi_c > 0$ they attract each other. In this case, $\xi_c > 0$ for the *SP* model and $\xi_c < 0$ for the *SG* model.

5.4.2 Experimental Results

This method has been tested on both real and synthetic images. 320 images (corresponding to 320 different genes) were randomly selected from the ABA within subsets of representative samples of all possible hippocampi, featuring both good-quality and low-quality images, with different characteristics. Moreover, 20 synthetic images have been created which represent simplified versions of the real ones. In these images, the hippocampus is made up of small circles having random radius and color; small and big ellipses were also added trying to simulate cells and, finally, gaussian and salt and pepper noise were introduced to add fuzziness to the images.

In [106], DE proved to be better than the other MHs used as comparison (PSO, GA, Scatter Search, Simulated Annealing and Levenberg-Marquardt) according to the final fitness. The results of the localization process have been manually evaluated by dividing the outcomes into three quality classes (see Figure 5.10):

1. Perfect Match: all points of the two models are over the corresponding parts and cover them almost entirely;

2. Good Localization: (i) all points of the two models belong to the regions which must be detected, but they do not cover them entirely or (ii) at most two points are slightly outside of them;

3. Error: all other possibilities, from three or more misplaced points to models which are located in a completely different position of the brain.

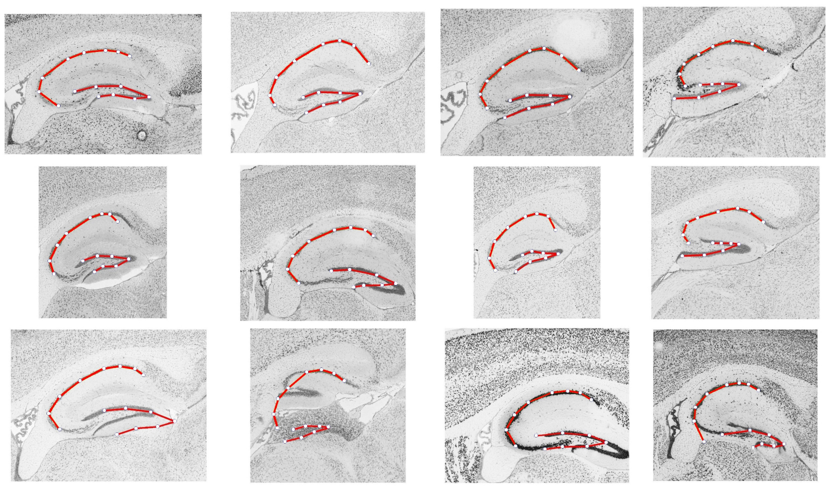


Figure 5.10: Results in the localization of the hippocampus. Upper row: perfect matches; middle row: good results; lower row: erroneous localizations.

This method was able to perfectly localize the hippocampus in 58.0% of synthetic images and in 47.8% of real images, and reached a good localization in 35% and 43.1% of cases, respectively. This means that this method was able to localize the hippocampus satisfactorily in 93.0% of the cases with synthetic images (20 images and 25 runs per image) and in 90.9% with real images (320 images and 1 run per image).

5.4.3 Automatic Parameter Tuning

To further improve the performance of the localization, SEPaT has been used to optimize DE's parameters [169]. Ten independent optimization processes have been run, 5 times using DE as tuner, 5 times using PSO. The parameters of the tuners were the same shown in Table 3.3, while the valid ranges of DE parameters are the ones which have been presented in Table 3.7. The function to be optimized in the tuning process was the maximization of the fitness function on a single hippocampus image, and the optimization was stopped after 100 DE generations.

Table 5.3 shows the manually-tuned parameters, which had been used originally, and the ones generated by SEPaT. Each row in the table presents the method used in the meta-optimization process and the parameter sets found. Two observations can be made:

- The parameter sets found by SEPaT in different runs show a very high homogeneity, proving once more the stability of such a method;
- DE population is very high compared to what has been reported in Chapter 3; the reason is that, in this case, the number of generations instead of the number of evaluations has been taken into account (because of the GPU-based implementations, all evaluations of a generation can be run in parallel), therefore more particles can be used without imposing any additional overhead.

Each set of parameters have been tested 10 times over the 320 hippocampus images. In order to check the statistical significance of the results obtained, a Wilcoxon signed-rank test was used to assess the statistical significance of the difference between the automatically-generated optimizers and the reference, manually-tuned ones (Original) for the two parts that compose the model (*SP* and *SG*), which proved that the quality of the SEPaT-generated parameters was enhanced. The parameter sets

Table 5.3: The manually-tuned parameters and the ten sets generated by SEPaT. The two columns for *SP* and *SG* indicate the average fitnesses and standard deviations. This is a maximization problem: a higher value represents a better solution.

	Parameters					*SP*		*SG*	
	CR	*F*	*Pop.Size*	*Mut*	*Cross*	Avg	Std	Avg	Std
Original	0.9	0.7	150	*TTB*	*Exp*	142.6	14.7	136.2	17.8
DE 1	0.859	0.41	121	*Rand*	*Bin*	144.5	12.1	141.0	13.9
DE 2	0.952	0.427	150	*Rand*	*Exp*	145.0	11.3	141.9	13.2
DE 3	0.952	0.419	150	*Rand*	*Exp*	145.0	11.5	141.8	13.1
DE 4	0.9	0.431	144	*Rand*	*Bin*	144.8	11.7	141.3	13.8
DE 5	0.954	0.44	115	*Rand*	*Exp*	144.9	11.7	141.4	13.7
PSO 1	0.949	0.448	149	*Rand*	*Exp*	145.1	11.2	141.8	13.1
PSO 2	0.953	0.471	143	*Rand*	*Exp*	145.0	11.3	141.7	13.2
PSO 3	0.974	0.473	150	*Rand*	*Exp*	145.0	11.5	142.0	13.0
PSO 4	0.783	0.347	150	*Rand*	*Bin*	144.6	11.8	141.0	13.7
PSO 5	0.922	0.437	139	*Rand*	*Exp*	145.0	11.2	141.4	13.2

automatically selected by SEPaT always lead to higher mean and lower standard deviation than the ones set after a time-consuming manual tuning.

5.5 Point Cloud Localization

In this last application we consider a system which is part of an architecture whose goal is to help users program robotic tasks [171]. To reach this goal, a subsystem for object recognition is required (see Figure 5.11). It receives input data

More details about the experiments presented in this section can be found in

- R. Ugolotti, G. Micconi, J. Aleotti, and S. Cagnoni. "GPU-based Point Cloud Recognition using Evolutionary Algorithms". In: Applications of Evolutionary Computation, Lecture Notes in Computer Science, pages 489-500. Springer, 2014
- R. Ugolotti and S. Cagnoni. "Multi-objective Parameter Tuning for PSO-based Point Cloud Localization". In: Advances in Artificial Life and Evolutionary Computation, Volume 445 of Communications in Computer and Information Science, pages 75-85. Springer, 2014

from a laser scanner mounted on the wrist of a six-degrees of freedom robot arm. The estimated accuracy of the whole measurement chain is about 1.5 cm, and the main sources of error are the variable remission of objects and the angle of incidence of the laser. Data undergo several preprocessing steps to refine the acquisition and are then passed to the FPFH (Fast Point Feature Histograms) based recognizer, along with a list of models stored in a database. The output of the recognizer indicates which objects are present in the scene and their pose. A description of a preliminary version of this system can be found in [126].

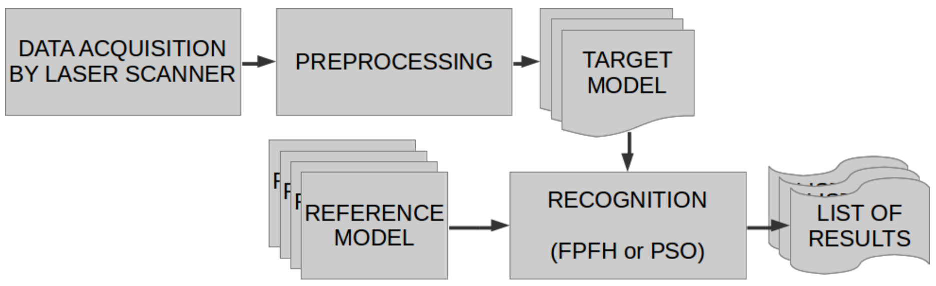


Figure 5.11: Representation of the system within which the FPFH recognizer, or the one based on PSO as an alternative, is used.

The main goal of this work is to recognize the pose of a known object, so the first step just consists of reading a point cloud from a database of available models. Since the model can only be subject to a rigid transformation, the search space is defined only by six degrees of freedom (translations and rotations around the three axes). This means that the dimensionality of the search space in which DE and PSO operate is six.

The scope of these experiments is to assess the performance of the bio-inspired model-based object recognition method in several situations and compare their results to those obtained using FPFH features (see next section). Figure 5.11 shows that this recognizer can be easily embedded into the existing system. Eventually, EMOPaT will be employed to show how it can further improve the performance of such a system.

5.5.1 FPFH

Fast Point Feature Histograms [144] are pose-invariant local features which represent the underlying surface model properties for all the elements composing a point cloud. These descriptors are computed for each point of a given point cloud and are generated by comparing the normal of a point with the normals of the points within a certain radius. For a more detailed description, please refer to [145]. Once all descriptors of the two point clouds (target and reference) have been computed, a particular version of the RANSAC algorithm (RANdom SAmple Consensus) [41] is used to find a raw alignment between the clouds. This version is called SAC-IA (SAmple Consensus - Initial Alignment) and is followed by a second step, which attempts to refine the previous alignment, using the Iterative Closest Point algorithm. Eventually, the two transformations found by the algorithms are composed in order to compute the full transformation needed to align the two clouds.

5.5.2 Fitness Function

This section describes the fitness function used by PSO and DE, as well as the system's GPU-based implementation. From now on, reference will be made only to a PSO-based implementation, but DE plays the same role.

The fitness function optimized by PSO is relatively straightforward. The target cloud T to be recognized (composed of N_T points) is compared with a reference cloud R extracted from a database, composed of N_R points. This reference is subject to a transformation M encoded by a PSO particle, to obtain $R' = M(R)$. The fitness of a particle is the average of the minimum distances of each point of T to the closest point of the roto-translated reference R'. More formally:

$$F(T,R') = \frac{1}{N_T} \sum_{p \in T} \min_{q \in R'} \left(dist(p,q) \right)$$

where $dist()$ is a valid distance metric between points; in this case the squared euclidean distance is used.

Each point cloud is expressed within a local reference frame centered around its

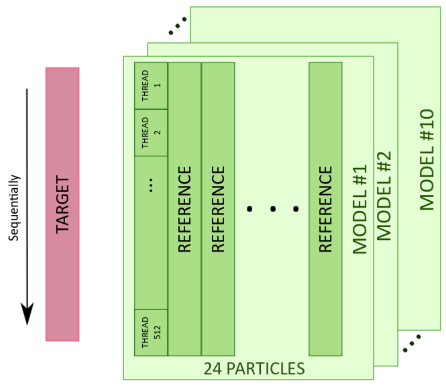


Figure 5.12: Scheme of the implementation of the fitness function on CUDA. Target points are computed sequentially. The parallel implementation relies on the fact that each point is compared (potentially) in parallel to all reference models (10 in this case), where each of the 24 PSO particles represents a possible transformation; 512 points are processed simultaneously for each particle.

centroid. A model can do a full rotation around each axis while the range of translation is limited to 10 cm in each direction, which is good enough to satisfy the requirements of the environment considered.

5.5.3 GPU Implementation

The entire system, including the computation of the fitness function, has been implemented on GPU. Several implementation designs have been tested. In the final one, two degrees of parallelism are exploited:

1. The i^{th} PSO particle represents a possible transformation M_i of the reference R and relies on a CUDA *block*, so all M_is can be computed in parallel;

2. Within each particle (so, within each *block*), each of many parallel threads processes a limited number of points of R, by firstly computing a portion of the transformed point cloud R' and then comparing it with all points of T.

The points of T are actually processed sequentially, but a significant speed-up can be obtained anyway because each of them is compared at the same time to several points of the reference cloud, and to different transformations of R. If the target is compared with more than one reference (for instance, to recognize which object has been scanned), a further level of parallelism can be added: several optimization processes can be executed in parallel using different reference models. Figure 5.12 outlines how the work is subdivided among CUDA *blocks* and *threads*.

The PSO and DE parameters (unless specified otherwise) were set as in Table 5.4. They have been chosen by manually generating 40 possible combinations, and testing them on the problem described in the next subsection. The configuration that gave the best average fitness was finally selected. Two PSO versions (with different topologies) proved to be almost equivalent.

Table 5.4: Manually-selected parameters used by DE and PSO.

DE	PSO_r	PSO_g
$CR = 0.9$	$c_1 = 1.19$	$c_1 = 1.8$
$F = 0.5$	$c_2 = 1.19$	$c_2 = 0.7$
Exponential Crossover	$w = 0.5$	$w = 0.72$
Target-to-best Mutation	Ring Topology	Global Topology
Population Size = 24	Population Size = 24	Population Size = 24
Generations = 90	Generations = 90	Generations = 90

In the tests that follow, except the last one, the same model (a wooden mallet) has been used as target and as reference, with random roto-translations applied to the target. Therefore, it was actually possible to achieve a perfect matching if the recognition process identified the correct transformation. The error in the localization process has been defined in terms of translation (euclidean distance between the translation obtained at the end of the experiment and the translation actually applied to the target) and rotation (angle between the estimated rotation and the one applied to the target).

5.5.4 Computation Time Comparison

We tested different PSO, DE and FPFH parameters (varying the number of generations in the first two, of RANSAC and ICP iterations for the other) in order to see how they behave when different time budgets are allowed. Four different time limits were considered: 0.7 s, 1.3 s, 2.3 s and 3.2 s. Figure 5.13 shows that FPFH reaches good results very quickly, but is unable to improve them any further, while MHs use their exploitation abilities to constantly refine their results. This is confirmed by statistical tests (Friedman test with the Dunn-Sidak correction, $p < 0.01$) which show that, within the first two time limits, FPFH is statistically better than the other methods considering translation and rotation errors, while this difference disappears with higher computation time budgets.

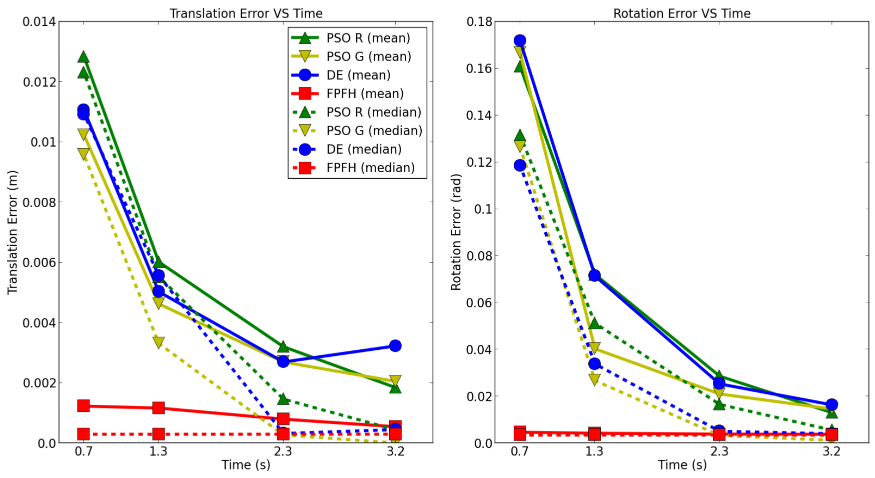


Figure 5.13: Error versus processing time allowed for optimization, computed over 100 experiments. Solid lines represent average values, while dotted lines represent medians.

Moreover, PSO/DE have usually a lower median and higher average than FPFH. This result (that will be confirmed in all other tests) proves that MHs have a better ability of finding more precise solutions, but they sometimes fall into local minima and fail to localize the object. On the contrary, FPFH steadily obtains good results, though worse than the ones obtained in the successful runs of the metaheuristics.

The sequential single-thread CPU implementation of the PSO recognizer takes an average of 60.5 s for 90 generations, which means it is 18.9 times slower than the GPU version on our PC (8-cores Intel Core i7 running at 3.40 GHz equipped with an nVIDIA GeForce GTX680 with 1536 cores working at 1.20 GHz). If the optimization process is parallelized over the 8 cores available on the CPU, the time needed is reduced to 16.4 s, thus the GPU is still 5.1 times faster.

5.5.5 Noise and Occlusions

In this section, we simulated some situations that can hamper object recognition, like noise and occlusions. The former was simulated by adding to each point of T a random value from a uniform distribution (we chose ranges of 0.001, 0.002, 0.005, 0.01 m), and the latter by removing all points above a certain percentile along a given dimension (we "occluded" 20%, 40%, 60% and 80% of the target). Figure 5.14 shows that FPFH is less robust to this kind of problems than PSO. Starting from an occlusion percentage of 60%, and for a noise level of 0.01 m, FPFH is significantly worse than all the MHs.

5.5.6 Object Recognition

After assessing the performance of these two methods under different conditions, we performed some tests on the problem of recognizing the object. In this case, the goal was not only to understand where the object was located, but also to recognize which is the target object, within a set of ten references: the wooden mallet used so far, a ewer, a burner, a toy horse, a mug and five boxes of different shapes and sizes. We performed 50 independent tests in which each object was used as target and compared to all the others both under normal conditions and simulating the presence of noise and occlusions. Results are presented in Figure 5.15.

The main conclusions can be summarized as follows:

- FPFH reaches good results in a very short time, but it is not able to further

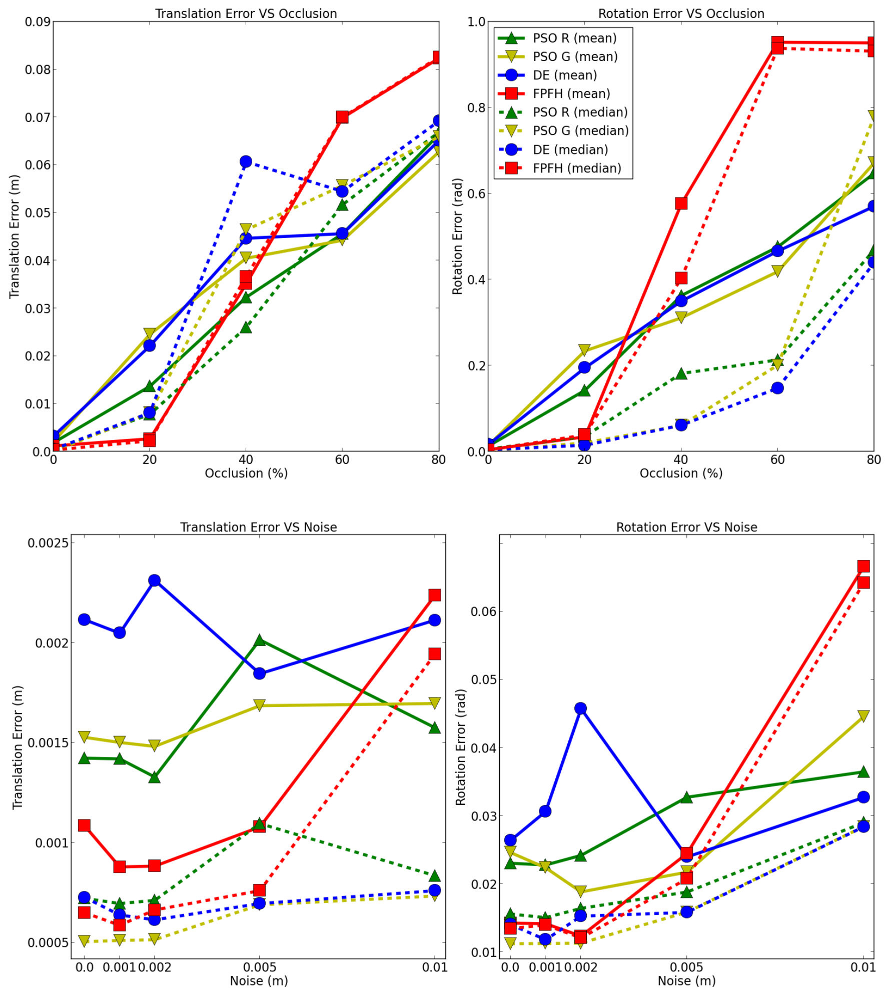


Figure 5.14: Variation of errors in the presence of occlusions (top) and noise added to the target (bottom) over 100 experiments. Solid lines represents average values, while dotted lines represents medians.

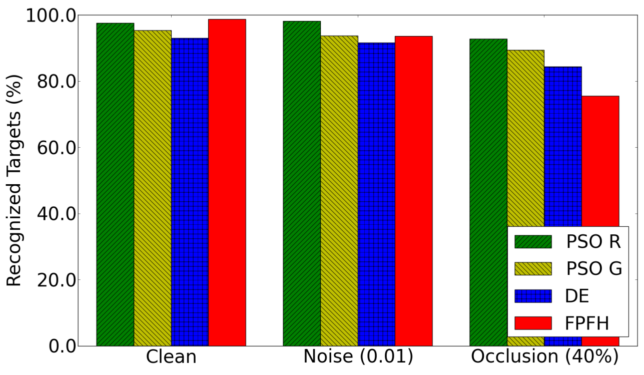


Figure 5.15: Percentage of correct recognitions over 500 experiments (50 repetitions for each of the ten different objects) for each entry of the bar chart. Again, one can see how FPFH performance degrades in the presence of occlusions.

improve them. Vice versa, the longer the time allowed to run MHs, the better the results they obtain;

- FPFH reaches good results almost always in ideal conditions, while MHs are able to achieve higher precision most of the times, but sometimes fail;
- MHs are more robust to noise and occlusions than FPFH.

5.5.7 Automatic Parameter Tuning

The goal of this test is to find PSO parameter sets that are able to find a good alignment between point clouds in a short time. For this reason, the two objectives EMOPaT tries to optimize are [167]:

1. Minimum time required to reach a fitness value of 0.1, averaged over 10 repetitions;
2. Best fitness reached after a time limit of 6 seconds, averaged over 10 repetitions.

NSGA-II was used as tuner with these parameters: 60 individuals, 30 generations, mutation rate = 0.125, crossover rate = 0.9. PSO parameters were allowed to vary within the range shown in Table 3.7. The goal was to estimate the localization of the wooden mallet, just like in the previous experiments.

Figure 5.16 shows the results obtained by EMOPaT. Each point represents the result of a set of parameters on the two objectives. The ones highlighted in red are the non-dominated ones. The main difference that can be observed between parameter sets that converge quickly to a solution with respect to the ones that reach a high final precision is that the former have a smaller population. This confirms the results obtained on benchmark functions in the previous chapters; small populations are good at reaching quickly a good fitness value, but generally fail in refining it because they are more likely to get stuck into local minima. Nevertheless, good populations are usually smaller than the ones usually suggested by common rules of thumb.

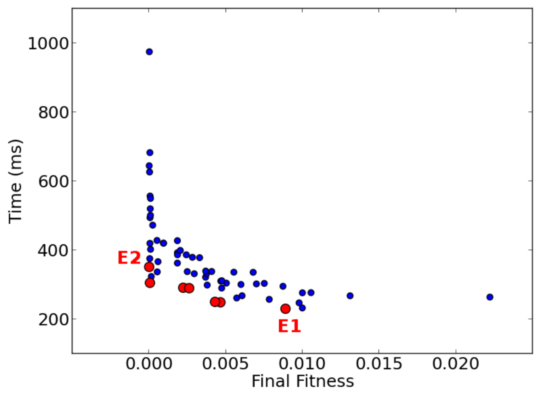


Figure 5.16: Results of EMOPaT. Larger, red dots represent the solutions approximating the Pareto front. E_1 and E_2 indicate the “fastest” and the “most precise” solutions, respectively.

We selected the two solutions which lie on opposite ends of the Pareto Front approximation (E_1 and E_2 in Figure 5.16, their parameter values are reported in Table 5.5), and tested them more deeply on the point-cloud alignment problem using each set of parameters with different time limits (0.3, 0.7, 1.3, 2.3 and 3.2 seconds) comparing them with the manually-tuned PSOs.

Table 5.5: PSO instances optimized by EMOPaT.

Parameter	E_1	E_2
Population Size	15	22
w	0.6652	0.5944
c_1	1.0479	2.1320
c_2	0.4271	0.7120
Topology	*Global*	*Global*

Table 5.6 shows, for each PSO configuration and for each time limit, the average fitness and standard deviation computed over 100 independent repetitions. The Wilcoxon signed-rank test ($p < 0.01$) has been performed between results at each time limit to see if there were significant differences; for each time limit, the best-performing PSO configurations are highlighted in bold.

Table 5.6: Average and standard deviation of fitness. Each column shows data of a PSO version, each row shows all data obtained at the corresponding time limit. PSO versions that perform statistically better are highlighted in bold.

PSO $\rightarrow$	PSO_g		PSO_r		E_1		E_2	
Time $\downarrow$	Avg	Std	Avg	Std	Avg	Std	Avg	Std
0.3 s	1.62e-01	9.34e-02	1.24e-01	5.89e-02	**7.66e-02**	**9.11e-02**	1.10e-01	1.12e-01
0.7 s	4.37e-02	2.96e-02	3.65e-02	2.41e-02	**2.48e-02**	**2.64e-02**	**2.05e-02**	**1.54e-02**
1.3 s	1.80e-02	1.55e-02	1.41e-02	2.61e-02	1.68e-02	2.01e-02	**9.30e-03**	**1.62e-02**
2.3 s	**7.48e-03**	**7.83e-03**	**7.64e-03**	**1.24e-02**	1.30e-02	1.44e-02	**6.65e-03**	**1.42e-02**
3.2 s	**4.02e-03**	**6.36e-03**	**5.65e-03**	**1.04e-02**	1.29e-02	1.25e-02	**4.53e-03**	**8.62e-03**

These results prove the correctness of the multi-objective approach. The set of parameters E_1, which was the fastest to reach the fitness goal, is in fact the best-performing one after 0.3 seconds, but it is not able to further improve its results. At the end of evolution, E_2 is comparable to the two manually-tuned ones. The main difference between E_2 and the latter is clear if one considers the intermediate time limits, on which E_2 performs better than PSO_g and PSO_r (see Table 5.4). This is a clear advantage of using EMOPaT. Eventually, many parameter combinations are able to solve the problem when a sufficient amount of time is given. EMOPaT has the ability to find, among all these possible solutions, the ones that are also able to reach a good fitness as fast as possible, because it also optimizes the other objective, that is not taken into consideration in a single-objective optimization or is difficult to consider during a manual tuning. Figure 5.17 shows the same results considering translation and rotation errors instead of fitness.

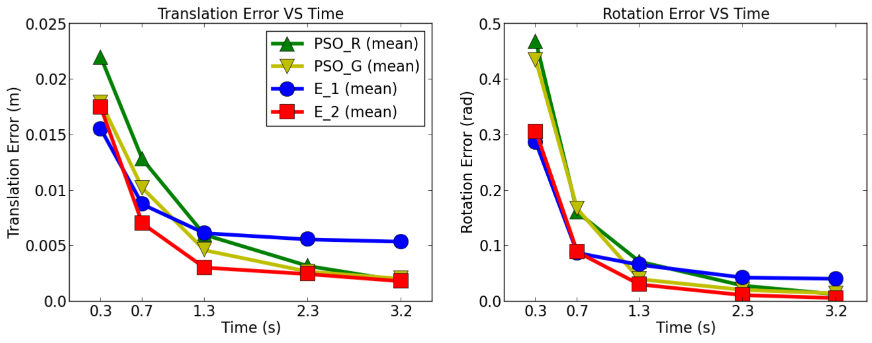


Figure 5.17: Results of the four PSOs with different time budgets.

5.6 Conclusions

This Chapter has described four applications in which the combination of bio-inspired techniques and model-based methods was used to successfully recognize objects in images and videos. This paradigm proved to be robust and of wide applicability. Automatic parameter tuning techniques were successfully employed to further improve the performance of the systems.

Chapter 6

Conclusions

An expert is a man who has made all the mistakes
which can be made in a very narrow field

– Niels Bohr

Several topics have been addressed in this thesis. It begins, in Chapter 2, with a description of several methods to automatically perform tuning and selection of the parameters of a metaheuristic of interest. Meta-Optimization has been chosen as a technique that combines easy implementation with high performance. In Chapter 3, a meta-optimization implementation called SEPaT (Simple Evolutionary Parameter Tuning) has been proposed. Comparison with a systematic search and with some other well-established methods which perform the same task have proven that SEPaT is an effective method to find good parameter configurations for a metaheuristic. Some of the main drawbacks of SEPaT, mostly its inability of finding validity ranges for the parameters and their role in affecting the algorithm's performance (shared with most of the meta-optimization techniques) induced us to extend it to the multi-objective paradigm. In doing so, our goal was to extract additional information regarding a metaheuristic applying the concept of *innovization*, which consists of finding a relationship between goals and parameters by analyzing the Pareto front. The multi-objective version presented in Chapter 4, called EMOPaT (Evolutionary Multi-Objective Parameter Tuning), is able to optimize the parameter values of a metaheuristic according to many objectives at the same time. The experiments show it is useful in many tasks, among which we can recall:

- finding good parameter values whose performances are indistinguishable from those obtained using its single-objective version;
- finding the range of validity for the values of a parameter;
- understanding the role and the utility (or the lack of it) of a parameter within an algorithm;
- performing an unbiased comparison between different metaheuristics, since they all undergo the same tuning procedure.

After that, in Chapter 5, we described the applications on which we have tested our meta-optimization algorithms: the recognition of objects in images and video sequences. Model-based approaches using bio-inspired optimization are a family of methods that were proved to have a wide applicability and achieve good performance in many different problems. Moreover, this thesis showed that automatic parameter tuning can improve the performance of object recognition without affecting its complexity, except for a small overhead in the design phase. In this field, meta-optimization can be considered as an operation that moves some of the complexity of the problem from the online phase to the design phase, simplifying the overall task.

The main direction towards which the work presented in this thesis may be expanded is the automation of the processes involved. Both in the analysis of EMOPaT's results and in the development of model-based object recognition methods, most work needs to be manually performed by the developer. In the former case, an automatic analysis of the Pareto front could be a significant improvement, especially if an investigation of the properties of the functions considered in the tuning process is also included. In the latter, at present, the developer has to define a model, a series of possible operations and a similarity function. These aspects are strictly dependent on the problem considered, but a more general framework can be studied, into which they can be included.

A Final Consideration

The work behind this thesis spans over more than three years. It actually started from what concluded this dissertation, the automatic recognition of objects in images and videos. When working on this approach I realized that the metaheuristics involved, although playing a fundamental role, were usually not used at their best. Therefore, an in-depth analysis showed me that this problem was not related only to the particular family of applications on which I was working on, and not only to the applications that rely on metaheuristics in general, but to the development itself of these techniques. As a result, I started to work mainly at understanding these algorithms, how they work and what is a good way to develop them in a more “scientific” way. Every year, many “novel” versions of bio-inspired algorithms are presented to the scientific community, by authors who claim that they are able to easily overtake the older versions; nonetheless the vast majority of this work is immediately forgotten. The main reason is that, following the “horse race” metaphor proposed by Johnson [68], winning a race is easy if you can accommodate the rules, choose the track and select your opponents (some of these problems have already been discussed at the beginning of Chapter 2). This is obviously not the first time this criticism have been made [37, 57] and many ideas have been proposed to tackle this problem [11, 47]. My idea is that automatic parameter tuning could be used as an effective technique to improve the quality of investigations on metaheuristics. In particular, the multi-objective approach proposed here, EMOPaT, has many properties that make it suitable for this task. EMOPaT is able to provide much information that a developer can use to widen its knowledge about the algorithm he/she is working on, such as whether a parameter is useful (or useless), the role of a parameter in an algorithm and so on. Moreover, as demonstrated in Section 4.1.1, EMOPaT is able to “understand” how a setting performs when compared to the “standard” ones, therefore it can automatically reject one that is not well-performing. Finally, using EMOPaT, it is possible to run an algorithm at its best, therefore it allows one to perform an unbiased comparison between different algorithms, as shown in Section 4.1.4.

Appendix A

libCudaOptimize

In this Appendix, the main concepts and advantages of GPGPU programming will be presented, followed by a basic introduction to *libCudaOptimize*, the GPU-based library we developed and which served as the basis for most of the code written for this work. The library is freely available on SourceForge[1].

A.1 CUDA Programming Model

General-purpose programming on GPU (GPGPU [129]) is the way of using a graphic card, which typically handles computations only for computer graphics and gaming, to execute applications traditionally managed by the Central Processing Unit (CPU). In this context, a GPU can be seen as a highly parallel, multi-threaded, many-core processor. Figure A.1 shows the theoretical number of floating-point operations per second for recent CPUs and GPUs.

CUDA™ (Compute Unified Distributed Architecture) is a GPGPU environment that includes a parallel computing architecture and programming model, developed by nVIDIA™ to exploit the massively parallel computation capabilities of its GPUs.

nVIDIA CUDA-C [125] is an extension of the C language for the development of GPUs routines (called *kernels*) that are executed N times in parallel by N uniquely

[1] http://sourceforge.net/projects/libcudaoptimize

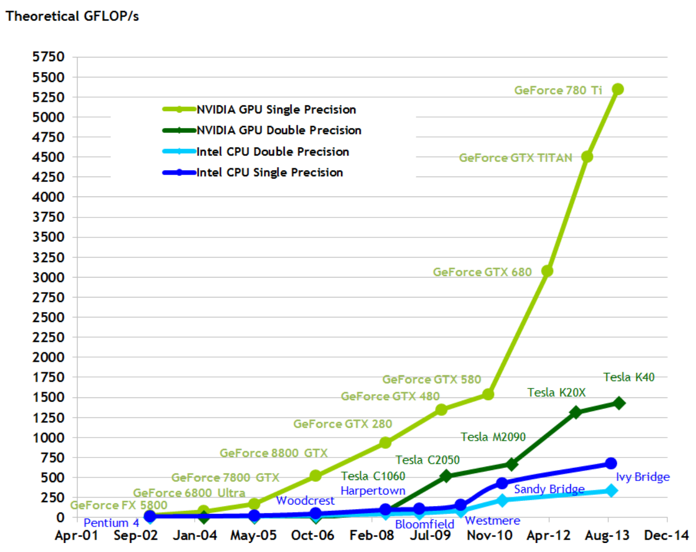


Figure A.1: Floating-point operations per second for recent CPUs and GPUs [125].

identified CUDA threads, following the Single Instruction Multiple Thread model: each *kernel* runs the same code, but on different data. *Kernels* are run on the device (GPU) while the rest of the code runs on the host (CPU). The threads on which the *kernels* run may be grouped into blocks. A block can be seen as a group of threads that share the same information and can exploit fast, local memory instead of using the slow, although large and persistent, global one. In this way they can exploit inherent properties of GPUs such as fast shared memory, atomic data manipulation, and synchronization.

The most desirable properties for an application to fit the GPGPU paradigm are low memory requirements, high degree of parallelization, high arithmetic intensity (ratio between arithmetic and memory operations) and few interactions between independent processes. These properties are perfectly matched by bio-inspired techniques, which employ a high number of elements that perform the same operation at the same time with very little (or, sometimes, none at all) information to be shared among them.

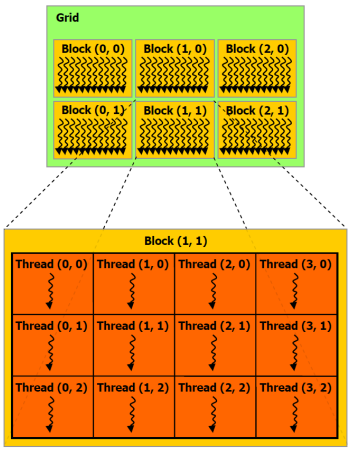


Figure A.2: Grid of Thread Blocks [125].

A.2 Bio-Inspired Algorithms on GPU

Many GPU implementations of BIAs have been presented in the last decade. One of the first was a GA implementation presented in 2005 [191]. Then, several improvements have been proposed to better exploit GPU properties such as hierarchical models [193], hybridization [183], extension to island models [137] and to multi-GPUs architectures [178]. The advantages of parallelizing GAs on GPUs have been recently investigated in [55].

The first implementations of PSO and DE relying on CUDA were developed in 2009 and 2010, respectively [29, 30]. After that, other implementations of DE have been developed [80, 199], and fast versions of PSO have been implemented by removing the constraint of synchronicity between particles [115]. The early PSO implementations on GPUs suffered from a coarse-grained parallelization that neglected the opportunity to compute the fitness function, usually the most time-consuming process, in parallel over the problem dimensions. This problem was solved in the PSO implementation described in [114] by adding a further level of parallelism. The early DE implementations contained similar inefficiencies, such as a partially sequential implementation of the fitness function and of the random number generator.

These problems were addressed by [80] using four kernels executed sequentially. This number has been reduced to three in [118].

Recently, Krömer et al have presented comprehensive reviews regarding GPU-based implementations of PSO [77], DE [78], and other MHs [79].

A.3 Overview of the library

The main idea behind the development of *libCudaOptimize* [118] is to offer a user the chance to apply MHs as simply and fast as possible to his own problem of interest, exploiting the parallelization opportunities offered by modern GPUs.

libCudaOptimize is a GPU-based open source library that allows users to run their methods in parallel to optimize a fitness function, introduce new optimization algorithms, or easily modify/extend existing ones. In the first case, the only thing one needs to do is to implement the new fitness function in C++ or CUDA-C, while in the second and third cases, one can take advantage of the framework offered by the library to avoid the need to go deep into basic implementation issues, especially as concerns parallel code.

Although no explicit understanding of CUDA-C or even of MHs is required, it is very useful anyway; nonetheless, one can use this library just by writing a C++ fitness function and launching one of the optimization techniques already implemented (to date PSO, DE and Scatter Search). This allows one to:

- implement commonly successful techniques with limited efforts;
- easily compare the results obtained by running different techniques on different functions;
- analyze the effects of changing values of the parameters which regulate the behavior of the optimization techniques on user-defined problems;
- run high-dimensional optimization experiments on consumer-level hardware, thanks to the efficient CUDA-C parallel implementation.

The documentation of *libcudaoptimize* can be found online[2] and some examples have been provided in [118]. In short, to implement a MH, one only needs to fill the functions that replicate the pseudo-algorithm presented in Algorithm 1:

- `initSolutions` initializes the population;
- `step` generates the population to be evaluated;
- `fitnessEvaluation` calls the fitness function;
- `update` selects the population to be passed to the next generation/iteration.

All the experiments presented in Chapter 5 have been implemented relying on this library. In the simplest cases, it has been enough to write a simple wrapper to allow the communication between the library and the already coded fitness function; in other cases the function has been ported from the original language (Matlab, C++) into CUDA-C; in the most complex cases, the functions of interest have been extracted from the library and embedded into the existing code. Figure A.3 shows the differences in terms of performance for the hippocampus localization when (i) both fitness function and MH were implemented in C++; (ii) fitness function was implemented in C++ using *libCudaOptimize*'s MHs; (iii) the entire program was run on GPU.

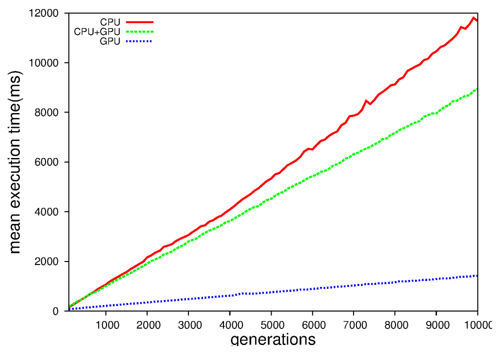


Figure A.3: Execution time versus number of generations.

[2] http://ibislab.ce.unipr.it/software/libcudaoptimize/doxygen

Bibliography

[1] B. Adenso-Díaz and M. Laguna. Fine-Tuning of Algorithms Using Fractional Experimental Designs and Local Search. *Operations Research*, 54(1):99–114, 2006.

[2] M. AlRashidi and M. El-Hawary. A Survey of Particle Swarm Optimization Applications in Electric Power Systems. *IEEE Transactions on Evolutionary Computation*, 13(4):913–918, 2009.

[3] C. Ansótegui, M. Sellmann, and K. Tierney. A gender-based genetic algorithm for the automatic configuration of algorithms. In *Principles and Practice of Constraint Programming*, pages 142–157. Springer, 2009.

[4] G. S. Atsalakis and K. P. Valavanis. Surveying stock market forecasting techniques - Part II: Soft computing methods. *Expert Systems with Applications*, 36(3, Part 2):5932 – 5941, 2009.

[5] A. Auger and B. Doerr. *Theory of Randomized Search Heuristics: Foundations and Recent Developments*, volume 1. World Scientific, 2011.

[6] A. Baak, M. Müller, G. Bharaj, H.-P. Seidel, and C. Theobalt. A Data-Driven Approach for Real-Time Full Body Pose Reconstruction from a Depth Camera. In *IEEE 13th International Conference on Computer Vision*, ICCV '13, pages 1092–1099, 2011.

[7] B. Babu and M. M. L. Jehan. Differential Evolution for multi-objective optimization. In *IEEE Congress on Evolutionary Computation*, volume 4 of *CEC'03*, pages 2696–2703, 2003.

[8] T. Bäck. Parallel Optimization of Evolutionary Algorithms. In *Parallel Problem Solving From Nature*, PPSN'94, pages 418–427, 1994.

[9] A. Banks, J. Vincent, and C. Anyakoha. A review of Particle Swarm Optimization. Part I: background and development. *Natural Computing*, 6(4):467–484, 2007.

[10] A. Barbier Accary and E. Galin. Fast distance Computation between a Point and Cylinders, Cones, Line-swept-Spheres and Cone-Spheres. *Journal of Graphics Tools*, 9(2):11–19, 2004.

[11] T. Bartz-Beielstein. *New experimentalism applied to evolutionary computation*. PhD thesis, Universität Dortmund, 2005.

[12] T. Bartz-Beielstein, C. Lasarczyk, and M. Preuß. Sequential parameter optimization. In *IEEE Congress on Evolutionary Computation*, volume 1 of *CEC'05*, pages 773–780, 2005.

[13] T. Bartz-Beielstein, P. Limbourg, J. Mehnen, K. Schmitt, K. E. Parsopoulos, and M. N. Vrahatis. Particle swarm optimizers for Pareto optimization with enhanced archiving techniques. In *IEEE Congress on Evolutionary Computation*, volume 3 of *CEC'03*, pages 1780–1787, 2003.

[14] L. Bezerra, M. López-Ibáñez, and T. Stützle. Automatic Design of Evolutionary Algorithms for Multi-Objective Combinatorial Optimization. In T. Bartz-Beielstein, J. Branke, B. Filipič, and J. Smith, editors, *Parallel Problem Solving from Nature - PPSN XIII*, volume 8672 of *Lecture Notes in Computer Science*, pages 508–517. Springer, 2014.

[15] M. Birattari, Z. Yuan, P. Balaprakash, and T. Stützle. F-Race and Iterated F-Race: An Overview. In T. Bartz-Beielstein, M. Chiarandini, L. Paquete,

and M. Preuß, editors, *Experimental Methods for the Analysis of Optimization Algorithms*, pages 311–336. Springer, 2010.

[16] C. Blum and A. Roli. Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison. *ACM Comput. Surv.*, 35(3):268–308, 2003.

[17] J. Branke and J. A. Elomari. Meta-optimization for Parameter Tuning with a Flexible Computing Budget. In *Genetic and Evolutionary Computation Conference*, GECCO'12, pages 1245–1252, 2012.

[18] J. Brest, S. Greiner, B. Boskovic, M. Mernik, and V. Zumer. Self-Adapting Control Parameters in Differential Evolution: A Comparative Study on Numerical Benchmark Problems. *IEEE Transactions on Evolutionary Computation*, 10(6):646–657, 2006.

[19] R. Brunelli. *Template Matching Techniques in Computer Vision: Theory and Practice*. Wiley Publishing, 2009.

[20] E. K. Burke, M. Hyde, G. Kendall, G. Ochoa, E. Özcan, and J. R. Woodward. A classification of hyper-heuristic approaches. In *Handbook of Metaheuristics*, pages 449–468. Springer, 2010.

[21] A. Carlisle and G. Dozier. An off-the-shelf PSO. In *Particle Swarm Optimization Workshop*, pages 1–6, 2001.

[22] P. Civicioglu and E. Besdok. A conceptual comparison of the Cuckoo-search, Particle Swarm Optimization, Differential Evolution and Artificial Bee Colony algorithms. *Artificial Intelligence Review*, 39(4):315–346, 2013.

[23] M. Clerc. *Particle swarm optimization*, volume 93. John Wiley & Sons, 2010.

[24] C. C. Coello, G. B. Lamont, and D. A. Van Veldhuizen. *Evolutionary Algorithms for solving multi-objective problems*. Springer, 2007.

[25] T. Cootes, G. Edwards, and C. Taylor. Active appearance models. In H. Burkhardt and B. Neumann, editors, *Computer Vision - ECCV'98*, volume 1407 of *Lecture Notes in Computer Science*, pages 484–498. Springer, 1998.

[26] T. Cootes, C. Taylor, D. Cooper, and J. Graham. Active Shape Models-Their Training and Application. *Computer Vision and Image Understanding*, 61(1):38 – 59, 1995.

[27] A. Czarn, C. MacNish, K. Vijayan, and B. Turlach. Statistical exploratory analysis of genetic algorithms: the importance of interaction. In *IEEE Congress on Evolutionary Computation*, volume 2 of *CEC'04*, pages 2288–2295 Vol.2, 2004.

[28] S. Das and P. Suganthan. Differential Evolution: A Survey of the State-of-the-Art. *IEEE Transactions on Evolutionary Computation*, 15:4–31, 2011.

[29] L. de Veronese and R. Krohling. Swarm's flight: Accelerating the particles using C-CUDA. In *IEEE Congress on Evolutionary Computation*, CEC'09, pages 3264–3270, 2009.

[30] L. de Veronese and R. Krohling. Differential Evolution algorithm on the GPU with C-CUDA. In *IEEE Congress on Evolutionary Computation*, CEC'10, pages 1–7, 2010.

[31] K. Deb, S. Bandaru, D. Greiner, A. Gaspar-Cunha, and C. C. Tutum. An integrated approach to automated innovization for discovering useful design principles: Case studies from engineering . *Applied Soft Computing*, 15(0):42 – 56, 2014.

[32] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multi-objective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002.

[33] K. Deb and A. Srinivasan. Innovization: Innovating design principles through optimization. In *Genetic and Evolutionary Computation Conference*, CEC'06, pages 1629–1636, 2006.

[34] J. Derrac, S. García, D. Molina, and F. Herrera. A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. *Swarm and Evolutionary Computation*, 1(1):3 – 18, 2011.

[35] J. Dréo. Using Performance Fronts for Parameter Setting of Stochastic Metaheuristics. In *Genetic and Evolutionary Computation Conference Companion: Late Breaking Papers*, GECCO'09, pages 2197–2200, 2009.

[36] F. Dressler and O. B. Akan. A survey on bio-inspired networking. *Computer Networks*, 54(6):881 – 900, 2010. New Network Paradigms.

[37] A. Eiben and M. Jelasity. A critical note on experimental research methodology in EC. In *IEEE Congress on Evolutionary Computation*, volume 1 of *CEC'02*, pages 582–587, 2002.

[38] A. Eiben, Z. Michalewicz, M. Schoenauer, and J. Smith. Parameter Control in Evolutionary Algorithms. In F. Lobo, C. Lima, and Z. Michalewicz, editors, *Parameter Setting in Evolutionary Algorithms*, volume 54 of *Studies in Computational Intelligence*, pages 19–46. Springer, 2007.

[39] A. E. Eiben and S. K. Smit. Parameter tuning for configuring and analyzing evolutionary algorithms. *Swarm and Evolutionary Computation*, 1(1):19 – 31, 2011.

[40] A. Engelbrecht. Particle Swarm Optimization: Global Best or Local Best? In *BRICS Congress on Computational Intelligence and 11th Brazilian Congress on Computational Intelligence*, BRICS-CCI CBIC'13, pages 124–135, 2013.

[41] M. A. Fischler and R. C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.

[42] C. A. Floudas and C. E. Gounaris. A Review of Recent Advances in Global Optimization. *J. of Global Optimization*, 45(1):3–38, 2009.

[43] B. Freisleben and M. Härtfelder. Optimization of Genetic Algorithms by Genetic Algorithms. In R. Albrecht, C. Reeves, and N. Steele, editors, *Artificial Neural Nets and Genetic Algorithms*, pages 392–399. Springer, 1993.

[44] R. Gämperle, S. D. Müller, and P. Koumoutsakos. A Parameter Study for Differential Evolution. In *Int. Conf. on Advances in Intelligent Systems, Fuzzy Systems, Evolutionary Computation*, pages 293–298, 2002.

[45] V. Ganapathi, C. Plagemann, S. Thrun, and D. Koller. Real Time Motion Capture Using a Single Time-of-Flight Camera. In *IEEE Conference on Computer Vision and Pattern Recognition*, CVPR'10, 2010.

[46] Y. Gao, Z. Ren, and C. Xu. A branch and bound-PSO hybrid algorithm for solving integer separable concave programming problems. *Applied Mathematical Sciences*, 1(11):517–525, 2007.

[47] S. García, D. Molina, M. Lozano, and F. Herrera. A study on the use of non-parametric tests for analyzing the evolutionary algorithms' behaviour: a case study on the CEC'2005 special session on real parameter optimization. *Journal of Heuristics*, 15(6):617–644, 2009.

[48] F. Glover. Future Paths for Integer Programming and Links to Artificial Intelligence. *Comput. Oper. Res.*, 13(5):533–549, 1986.

[49] J. Gratch and G. DeJong. COMPOSER: A Probabilistic Solution to the Utility Problem in Speed-up Learning. In *National Conference on Artificial Intelligence*, AAAI'92, pages 235–240, 1992.

[50] M. Greeff and A. Engelbrecht. Dynamic Multi-objective Optimisation Using PSO. In N. Nedjah, L. dos Santos Coelho, and L. de Macedo Mourelle, editors, *Multi-Objective Swarm Intelligent Systems*, volume 261 of *Studies in Computational Intelligence*, pages 105–123. Springer, 2010.

[51] J. Grefenstette. Optimization of control parameters for genetic algorithms. *IEEE Transactions on Systems, Man, and Cybernetics*, 16(1):122–128, 1986.

[52] E. Hart, P. Ross, and D. Corne. Evolutionary Scheduling: A Review. *Genetic Programming and Evolvable Machines*, 6(2):191–220, 2005.

[53] S. Haykin. *Neural networks and learning machines*, volume 3. Pearson Education Upper Saddle River, 2009.

[54] M. Heydarian, M. Noseworthy, M. Kamath, C. Boylan, and W. Poehlman. Optimizing the Level Set Algorithm for Detecting Object Edges in MR and CT Images. *IEEE Transactions on Nuclear Science*, 56(1):156–166, 2009.

[55] J. Hofmann, S. Limmer, and D. Fey. Performance investigations of genetic algorithms on graphics cards. *Swarm and Evolutionary Computation*, 12(0):33 – 47, 2013.

[56] J. H. Holland. *Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence*. U Michigan Press, 1975.

[57] J. N. Hooker. Testing heuristics: We have it all wrong. *Journal of Heuristics*, 1(1):33–42, 1995.

[58] H. Hoos. Automated Algorithm Configuration and Parameter Tuning. In Y. Hamadi, E. Monfroy, and F. Saubion, editors, *Autonomous Search*, pages 37–71. Springer, 2012.

[59] R. Horst and H. E. Romeijn. *Handbook of global optimization*, volume 2. Springer, 2002.

[60] E. Hruschka, R. Campello, A. Freitas, and A. de Carvalho. A Survey of Evolutionary Algorithms for Clustering. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 39(2):133–155, 2009.

[61] X. Huang and D. Metaxas. Metamorphs: Deformable Shape and Appearance Models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(8):1444–1459, 2008.

[62] F. Hutter, H. H. Hoos, K. Leyton-Brown, and T. Stützle. ParamILS: An Automatic Algorithm Configuration Framework. *Journal of Artificial Intelligence Research*, 36(1):267–306, 2009.

[63] F. Hutter, H. H. Hoos, and T. Stützle. Automatic algorithm configuration based on local search. In *National Conference on Artificial Intelligence*, volume 7 of *AAAI'07*, pages 1152–1157, 2007.

[64] A. Jain, Y. Zhong, and S. Lakshmanan. Object matching using deformable templates. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(3):267–278, 1996.

[65] J. Jaramillo, M. Orozco, G. Castellanos, and J. Riano. Variational Shape Model for Lip Postures Recognition Using GA. In *Electronics, Robotics and Automotive Mechanics Conference*, volume 1, pages 25–29, 2006.

[66] W. Ji and K. Wang. An improved Particle Swarm Optimization algorithm. In *International Conference on Computer Science and Network Technology*, volume 1 of *ICCSNT'11*, pages 585–589, 2011.

[67] Y. Jin and J. Branke. Evolutionary optimization in uncertain environments-a survey. *IEEE Transactions on Evolutionary Computation*, 9(3):303–317, 2005.

[68] D. S. Johnson. A theoretician's guide to the experimental analysis of algorithms. *Data structures, near neighbor searches, and methodology: fifth and sixth DIMACS implementation challenges*, 59:215–250, 2002.

[69] T. Kailath. The Divergence and Bhattacharyya Distance Measures in Signal Selection. *IEEE Transactions on Communication Technology*, 15(1):52–60, 1967.

[70] Y.-T. Kao and E. Zahara. A hybrid genetic algorithm and Particle Swarm Optimization for multimodal functions. *Applied Soft Computing*, 8(2):849–857, 2008.

[71] M. Kass, A. Witkin, and D. Terzopoulos. Snakes: Active contour models. *Int J of Computer Vision*, 1:321–331, 1988.

[72] J. Kennedy and M. Clerc. Standard PSO 2006, 2006. Online: http://www.particleswarm.info/Standard_PSO_2006.c.

[73] J. Kennedy and R. Eberhart. Particle Swarm Optimization. In *IEEE International Conference on Neural Networks*, volume 4 of *ICNN'95*, pages 1942–1948, 1995.

[74] R. Kicinger, T. Arciszewski, and K. D. Jong. Evolutionary Computation and Structural Design: A Survey of the State-of-the-art. *Comput. Struct.*, 83(23-24):1943–1978, 2005.

[75] B.-I. Koh, A. D. George, R. T. Haftka, and B. J. Fregly. Parallel asynchronous Particle Swarm Optimization. *International Journal for Numerical Methods in Engineering*, 67(4):578–595, 2006.

[76] J. R. Koza. *Genetic programming: on the programming of computers by means of natural selection*, volume 1. MIT press, 1992.

[77] P. Krömer, J. Platoš, and V. Snášel. A brief survey of advances in Particle Swarm Optimization on Graphic Processing Units. In *IEEE World Congress on Nature and Biologically Inspired Computing*, NaBIC'13, pages 182–188, 2013.

[78] P. Krömer, J. Platoš, and V. Snášel. A brief survey of Differential Evolution on Graphic Processing Units. In *Symposium on Differential Evolution*, SDE, pages 157–164, 2013.

[79] P. Krömer, J. Platoš, and V. Snášel. Nature-Inspired Meta-Heuristics on Modern GPUs: State of the Art and Brief Survey of Selected Algorithms. *Int. J. Parallel Program.*, 42(5):681–709, 2014.

[80] P. Krömer, V. Snášel, J. Platoš, and A. Abraham. Many-threaded implementation of Differential Evolution for the CUDA platform. In *Genetic and Evolutionary Computation Conference*, GECCO'11, pages 1595–1602, 2011.

[81] R. Kulkarni and G. Venayagamoorthy. Particle Swarm Optimization in Wireless-Sensor Networks: A Brief Survey. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 41(2):262–267, 2011.

[82] J. Lampinen and I. Zelinka. On stagnation of the Differential Evolution algorithm. In *Proceedings of MENDEL*, pages 76–83, 2000.

[83] P. Larrañaga and J. A. Lozano. *Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation*. Kluwer Academic Publishers, 2001.

[84] E. S. Lein, M. J. Hawrylycz, N. Ao, M. Ayres, A. Bensinger, A. Bernard, A. F. Boe, M. S. Boguski, K. S. Brockway, E. J. Byrnes, et al. Genome-wide atlas of gene expression in the adult mouse brain. *Nature*, 445(7124):168–176, 2006.

[85] R. Lewis. A survey of metaheuristic-based techniques for university timetabling problems. *OR Spectrum*, 30(1):167–190, 2008.

[86] K. Leyton-Brown, E. Nudelman, G. Andrew, J. Mcfadden, and Y. Shoham. A Portfolio Approach to Algorithm Selection. In *International Joint Conference on Artifical Intelligence*, IJCAI'03, pages 1542–1543, 2003.

[87] Z. Li-ping, Y. Huan-jun, and H. Shang-xu. Optimal choice of parameters for Particle Swarm Optimization. *Journal of Zhejiang University Science A*, 6(6):528–534, 2005.

[88] J. Liang, B. Qu, and P. Suganthan. Problem definitions and evaluation criteria for the CEC 2013 special session on real-parameter optimization. Technical Report 201212, Computational Intelligence Laboratory, Zhengzhou University, 2013.

[89] J. Liang and P. Suganthan. Dynamic multi-swarm particle swarm optimizer. In *Proceedings of Swarm Intelligence Symposium*, pages 124–129, 2005.

[90] T. Liao and T. Stützle. Benchmark results for a simple hybrid algorithm on the CEC 2013 benchmark set for real-parameter optimization. In *IEEE Congress on Evolutionary Computation*, CEC'13, pages 1938–1944, 2013.

[91] T. W. Liao. Two hybrid Differential Evolution algorithms for engineering design optimization. *Applied Soft Computing*, 10(4):1188 – 1199, 2010. Optimisation Methods & Applications in Decision-Making Processes.

[92] F. Lobo, C. Lima, and Z. Michalewicz. *Parameter Setting in Evolutionary Algorithms*, volume 54 of *Studies in Computational Intelligence*. Springer, 2007.

[93] M. A. Lones. Metaheuristics in Nature-inspired Algorithms. In *Genetic and Evolutionary Computation Conference Companion*, GECCO'14, pages 1419–1422, 2014.

[94] M. López-Ibánez, J. Dubois-Lacoste, T. Stützle, and M. Birattari. The irace package, iterated race for automatic algorithm configuration. Technical Report TR/IRIDIA/2011-004, IRIDIA, UniversitÃl' Libre de Bruxelles, Belgium, 2011.

[95] S. Luke and A. K. A. Talukder. Is the meta-EA a Viable Optimization Method? In *Genetic and Evolutionary Computation Conference*, GECCO'13, pages 1533–1540, 2013.

[96] R. Mallipeddi and P. Suganthan. Empirical study on the effect of population size on Differential Evolution Algorithm. In *IEEE Congress on Evolutionary Computation*, CEC'08, pages 3663–3670, 2008.

[97] R. Mallipeddi and P. Suganthan. Ensemble Differential Evolution algorithm for CEC2011 problems. In *IEEE Congress on Evolutionary Computation*, CEC'11, pages 1557–1564, 2011.

[98] O. Maron and A. W. Moore. The Racing Algorithm: Model Selection for Lazy Learners. *Artificial Intelligence Review*, 11:193–225, 1997.

[99] J. L. F. Martínez and E. G. Gonzalo. The generalized PSO: A new door to PSO evolution. *J. Artif. Evol. App.*, 2008:5:1–5:15, 2008.

[100] U. Maulik. Medical Image Segmentation Using Genetic Algorithms. *IEEE Transactions on Information Technology in Biomedicine*, 13(2):166–173, 2009.

[101] P. Medici, C. Caraffi, E. Cardarelli, P. Porta, and G. Ghisio. Real time road signs classification. In *IEEE International Conference on Vehicular Electronics and Safety*, ICVES'08, pages 253–258, 2008.

[102] M. Meissner, M. Schmuker, and G. Schneider. Optimized Particle Swarm Optimization (OPSO) and its application to artificial neural network training. *BMC Bioinformatics*, 7, 2006.

[103] R. Mendes and A. S. Mohais. DynDE: a differential evolution for dynamic optimization problems. In *IEEE Congress on Evolutionary Computation*, volume 3 of *CEC'05*, pages 2808–2815, 2005.

[104] R. Mercer and J. Sampson. Adaptive search using a reproductive metaplan. *Kybernetes*, 7:215–228, 1978.

[105] P. Mesejo. *Automatic segmentation of anatomical structures using deformable models and bio-inspired/soft computing*. PhD thesis, Università degli Studi di Parma. Dipartimento di Ingegneria dell'Informazione, 2014.

[106] P. Mesejo, R. Ugolotti, F. Di Cunto, M. Giacobini, and S. Cagnoni. Automatic hippocampus localization in histological images using Differential Evolution-based deformable models. *Pattern Recognition Letters*, 34(3):299–307, 2013.

[107] P. Mesejo, A. Valsecchi, L. Marrakchi-Kacem, S. Cagnoni, and S. Damas. Biomedical image segmentation using geometric deformable models and metaheuristics. *Computerized Medical Imaging and Graphics*, in press, 2014.

[108] S. Mitra, S. Pal, and P. Mitra. Data mining in soft computing framework: a survey. *IEEE Transactions on Neural Networks*, 13(1):3–14, 2002.

[109] A. Mogelmose, M. Trivedi, and T. Moeslund. Vision-based traffic sign detection and analysis for intelligent driver assistance systems: Perspectives and survey. *IEEE Transactions on Intelligent Transportation Systems*, 13(4):1484–1497, 2012.

[110] E. Montero, M.-C. Riff, L. Pérez-Caceres, and C. Coello Coello. Are state-of-the-art fine-tuning algorithms able to detect a dummy parameter? In C. Coello, V. Cutello, K. Deb, S. Forrest, G. Nicosia, and M. Pavone, editors, *Parallel Problem Solving from Nature - PPSN XII*, volume 7491 of *Lecture Notes in Computer Science*, pages 306–315. Springer, 2012.

[111] P. W. Moore and G. K. Venayagamoorthy. Evolving digital circuits using hybrid Particle Swarm Optimization and Differential Evolution. *International Journal of neural systems*, 16(03):163–177, 2006.

[112] A. Mousa, M. El-Shorbagy, and W. Abd-El-Wahed. Local search based hybrid Particle Swarm Optimization algorithm for multiobjective optimization. *Swarm and Evolutionary Computation*, 3(0):1 – 14, 2012.

[113] L. Mussi, S. Cagnoni, E. Cardarelli, F. Daolio, P. Medici, and P. Porta. GPU implementation of a road sign detector based on Particle Swarm Optimization. *Evolutionary Intelligence*, 3(3):155–169, 2010.

[114] L. Mussi, F. Daolio, and S. Cagnoni. Evaluation of parallel Particle Swarm Optimization algorithms within the CUDA architecture. *Information Sciences*, 181:4642–4657, 2011.

[115] L. Mussi, Y. S. Nashed, and S. Cagnoni. GPU-based Asynchronous Particle Swarm Optimization. In *Genetic and Evolutionary Computation Conference*, GECCO'11, pages 1555–1562, 2011.

[116] V. Nannen and A. E. Eiben. Relevance Estimation and Value Calibration of Evolutionary Algorithm Parameters. In *International Joint Conference on Artifical Intelligence*, IJCAI'07, pages 975–980, 2007.

[117] Y. S. G. Nashed, P. Mesejo, R. Ugolotti, J. Dubois-Lacoste, and S. Cagnoni. A comparative study of three GPU-based metaheuristics. In *Parallel Problem Solving from Nature - PPSN XII*, pages 398–407. Springer, 2012.

[118] Y. S. G. Nashed, R. Ugolotti, P. Mesejo, and S. Cagnoni. libCudaOptimize: an open source library of GPU-based metaheuristics. In *Genetic and Evolutionary Computation Conference Companion*, GECCO'12, pages 117–124, 2012.

[119] A. L. Nelson, G. J. Barlow, and L. Doitsidis. Fitness functions in evolutionary robotics: A survey and analysis. *Robotics and Autonomous Systems*, 57(4):345 – 370, 2009.

[120] F. Neri and V. Tirronen. Recent advances in Differential Evolution: a survey and experimental analysis. *Artif. Intell. Rev.*, 33:61–106, 2010.

[121] Q. Ni and J. Deng. A new logistic dynamic Particle Swarm Optimization algorithm based on random topology. *The Scientific World Journal*, 2013, 2013.

[122] N. Noman and H. Iba. Accelerating Differential Evolution Using an Adaptive Local Search. *IEEE Transactions on Evolutionary Computation*, 12(1):107–125, 2008.

[123] K. A. Norman. How hippocampus and cortex contribute to recognition memory: revisiting the complementary learning systems model. *Hippocampus*, 20(11):1217–1227, 2010.

[124] J. Novo, M. Penedo, and J. Santos. Localisation of the optic disc by means of GA-optimised Topological Active Nets. *Image and Vision Computing*, 27(10):1572 – 1584, 2009. Special Section: Computer Vision Methods for Ambient Intelligence.

[125] nVIDIA Corporation. *nVIDIA CUDA Programming Guide v. 6.0*, 2014.

[126] F. Oleari, D. Lodi Rizzini, and S. Caselli. A low-cost stereo system for 3D object recognition. In *IEEE International Conference on Intelligent Computer Communication and Processing*, ICCP'13, pages 127–132, 2013.

[127] M. Oltean. Evolving Evolutionary Algorithms Using Linear Genetic Programming. *Evolutionary Computation*, 13:2005, 2005.

[128] I. Osman and G. Laporte. Metaheuristics: A bibliography. *Annals of Operations Research*, 63(5):511–623, 1996.

[129] J. D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Krüger, A. E. Lefohn, and T. J. Purcell. A Survey of General-Purpose Computation on Graphics Hardware. *Computer Graphics Forum*, 26:80–113, 2007.

[130] M. Paliwal and U. A. Kumar. Neural networks and statistical techniques: A review of applications. *Expert Systems with Applications*, 36(1):2 – 17, 2009.

[131] G. Pampara, A. P. Engelbrecht, and N. Franken. Binary Differential Evolution. In *IEEE Congress on Evolutionary Computation*, CEC'06, pages 1873–1879, 2006.

[132] Q.-K. Pan, M. F. Tasgetiren, and Y.-C. Liang. A discrete Differential Evolution algorithm for the permutation flowshop scheduling problem. *Computers & Industrial Engineering*, 55(4):795–816, 2008.

[133] W. Paszkowicz. Genetic algorithms, a nature-inspired tool: A survey of applications in materials science and related fields: Part II. *Materials and Manufacturing Processes*, 28(7):708–725, 2013.

[134] M. E. H. Pedersen. Tuning and Simplifying Heuristical Optimization. Master's thesis, University of Southampton, 2010.

[135] E. Pitzer and M. Affenzeller. A comprehensive survey on fitness landscape analysis. In J. Fodor, R. Klempous, and C. Suárez Araujo, editors, *Recent*

Advances in Intelligent Engineering Systems, volume 378 of *Studies in Computational Intelligence*, pages 161–191. Springer, 2012.

[136] R. Poli. Analysis of the publications on the applications of Particle Swarm Optimisation. *J. Artificial Evolution and Applications*, 2008:1–10, 2008.

[137] P. Pospichal, J. Jaros, and J. Schwarz. Parallel Genetic Algorithm on the CUDA Architecture. In C. Di Chio, S. Cagnoni, C. Cotta, M. Ebner, A. Ekàrt, A. Espárcia-Alcázar, C.-K. Goh, J. Merelo, F. Neri, M. Preuß, J. Togelius, and G. Yannakakis, editors, *Applications of Evolutionary Computation*, volume 6024 of *Lecture Notes in Computer Science*, pages 442–451. Springer, 2010.

[138] M. Preuß. Considerations of budget allocation for sequential parameter optimization (SPO). In *Workshop on Empirical Methods for the Analysis of Algorithms*, pages 35–40, 2006.

[139] A. Qin, V. Huang, and P. Suganthan. Differential Evolution Algorithm With Strategy Adaptation for Global Numerical Optimization. *IEEE Transactions on Evolutionary Computation*, 13(2):398–417, 2009.

[140] B. Qu, J. Liang, and P. Suganthan. Niching Particle Swarm Optimization with local search for multi-modal optimization. *Information Sciences*, 197(0):131 – 143, 2012.

[141] J. Rada-Vilela, M. Zhang, and W. Seah. Random Asynchronous PSO. In *International Conference on Automation, Robotics and Applications*, ICARA'11, pages 220–225, 2011.

[142] I. Rechenberg. *Evolutionsstrategie Optimierung technischer Systeme nach Prinzipien der biologishen Evolution*. Frommann-Holzboog, 1973.

[143] T. Robič and B. Filipič. DEMO: Differential Evolution for multiobjective optimization. In *International Conference on Evolutionary Multi-Criterion Optimization*, pages 520–533, 2005.

[144] R. B. Rusu, N. Blodow, and M. Beetz. Fast point feature histograms (FPFH) for 3D registration. In *IEEE International Conference on Robotics and Automation*, ICRA'09, pages 3212–3217, 2009.

[145] R. B. Rusu, Z. C. Marton, N. Blodow, and M. Beetz. Learning informative point classes for the acquisition of object model maps. In *IEEE International Conference on Control, Automation, Robotics and Vision*, ICARCV'08, pages 643–650, 2008.

[146] G. D. Ruxton. The unequal variance t-test is an underused alternative to student's t-test and the Mann–Whitney U test. *Behavioral Ecology*, 17(4):688–690, 2006.

[147] K. Safarzyńska and J. van den Bergh. Evolutionary models in economics: a survey of methods and building blocks. *Journal of Evolutionary Economics*, 20(3):329–373, 2010.

[148] R. Sano, T. Shindo, K. Jin'no, and T. Saito. PSO-based multiple optima search systems with switched topology. In *IEEE Congress on Evolutionary Computation*, CEC'12, pages 1–7, 2012.

[149] M. Settles and T. Soule. Breeding Swarms: A GA/PSO Hybrid. In *Genetic and Evolutionary Computation Conference*, GECCO'05, pages 161–168, 2005.

[150] X. Shi, Y. Liang, H. Lee, C. Lu, and L. Wang. An improved GA and a novel PSO-GA-based hybrid algorithm. *Information Processing Letters*, 93(5):255 – 261, 2005.

[151] Y. Shi and R. C. Eberhart. Empirical study of Particle Swarm Optimization. In *IEEE Congress on Evolutionary Computation*, volume 3 of *CEC'99*, 1999.

[152] S. K. Smit and A. E. Eiben. Comparing Parameter Tuning Methods for Evolutionary Algorithms. In *IEEE Congress on Evolutionary Computation*, CEC'09, pages 399–406, 2009.

[153] S. K. Smit and A. E. Eiben. Beating the 'world champion' evolutionary algorithm via REVAC tuning. In *IEEE Congress on Evolutionary Computation*, CEC'10, pages 1–8, 2010.

[154] S. K. Smit and A. E. Eiben. Parameter Tuning of Evolutionary Algorithms: Generalist vs. Specialist. In C. Di Chio, S. Cagnoni, C. Cotta, M. Ebner, A. Ekàrt, A. Espárcia-Alcázar, C.-K. Goh, J. Merelo, F. Neri, M. Preuß, J. Togelius, and G. Yannakakis, editors, *Applications of Evolutionary Computation*, volume 6024 of *Lecture Notes in Computer Science*, pages 542–551. Springer, 2010.

[155] S. K. Smit, A. E. Eiben, and Z. Szlávik. An MOEA-based Method to Tune EA Parameters on Multiple Objective Functions. In *International Joint Conference on Artifical Intelligence*, IJCCI'10, pages 261–268, 2010.

[156] K. Socha and M. Dorigo. Ant Colony Optimization for continuous domains. *European Journal of Operational Research*, 185(3):1155 – 1173, 2008.

[157] K. Sörensen. Metaheuristics - the metaphor exposed. *International Transactions in Operational Research*, 22:3–18, 2013.

[158] R. Storn and K. Price. Differential Evolution- A Simple and Efficient Adaptive Scheme for Global Optimization over Continuous Spaces. Technical report, International Computer Science Institute, 1995.

[159] P. N. Suganthan, N. Hansen, J. J. Liang, K. Deb, Y.-P. Chen, A. Auger, and S. Tiwari. Problem definitions and evaluation criteria for the CEC 2005 special session on real-parameter optimization. Technical report, Nanyang Technological University, Singapore, 2005.

[160] G. Taguchi, D. Clausing, and L. T. Watanabe. *System of experimental design: engineering methods to optimize quality and minimize costs*, volume 1. UNIPUB/Kraus International Publications, 1987.

[161] J. Taylor, J. Shotton, T. Sharp, and A. Fitzgibbon. The vitruvian manifold: inferring dense correspondences for one-shot human pose estimation. In *IEEE Conference on Computer Vision and Pattern Recognition*, CVPR'12, pages 103 –110, 2012.

[162] D. Terzopoulos. On Matching Deformable Models to Images: Direct and Iterative Solutions. In *Topical Meeting on Machine Vision*, volume 12, pages 160–167, 1987.

[163] D. Terzopoulos and K. Fleischer. Deformable models. *The Visual Computer*, 4:306–331, 1988.

[164] R. Ugolotti and S. Cagnoni. Design of EAs for Continuous Optimization aided by Multi-Objective Parameter Tuning. *Submitted to Evolutionary Computation*.

[165] R. Ugolotti and S. Cagnoni. Differential Evolution Based Human Body Pose Estimation from Point Clouds. In *Genetic and Evolutionary Computation Conference*, GECCO'13, pages 1389–1396, 2013.

[166] R. Ugolotti and S. Cagnoni. Analysis of Evolutionary Algorithms using multi-objective parameter tuning. In *Genetic and Evolutionary Computation Conference*, GECCO'14, pages 1343–1350, 2014.

[167] R. Ugolotti and S. Cagnoni. Multi-objective Parameter Tuning for PSO-based Point Cloud Localization. In C. Pizzuti and G. Spezzano, editors, *Advances in Artificial Life and Evolutionary Computation*, volume 445 of *Communications in Computer and Information Science*, pages 75–85. Springer, 2014.

[168] R. Ugolotti, P. Mesejo, S. Cagnoni, M. Giacobini, and F. Di Cunto. Automatic Hippocampus Localization in Histological Images Using PSO-based Deformable Models. In *Genetic and Evolutionary Computation Conference Companion*, GECCO'11, pages 487–494, 2011.

[169] R. Ugolotti, P. Mesejo, Y. S. G. Nashed, and S. Cagnoni. GPU-Based Automatic Configuration of Differential Evolution: A Case Study. In L. Correia, L. Reis, and J. Cascalho, editors, *Progress in Artificial Intelligence*, volume 8154 of *Lecture Notes in Computer Science*, pages 114–125. Springer, 2013.

[170] R. Ugolotti, P. Mesejo, S. Zongaro, B. Bardoni, G. Berto, F. Bianchi, I. Molineris, M. Giacobini, S. Cagnoni, and F. Di Cunto. Visual search of neuropil-enriched RNAs from brain in situ hybridization data through the image analysis pipeline hippo-ATESC. *PloS one*, 8(9), 2013.

[171] R. Ugolotti, G. Micconi, J. Aleotti, and S. Cagnoni. GPU-Based Point Cloud Recognition Using Evolutionary Algorithms. In A. I. Esparcia-Alcázar and A. M. Mora, editors, *Applications of Evolutionary Computation*, Lecture Notes in Computer Science, pages 489–500. Springer, 2014.

[172] R. Ugolotti, Y. S. G. Nashed, and S. Cagnoni. Real-Time GPU Based Road Sign Detection and Classification. In C. Coello, V. Cutello, K. Deb, S. Forrest, G. Nicosia, and M. Pavone, editors, *Parallel Problem Solving from Nature - PPSN XII*, volume 7491 of *Lecture Notes in Computer Science*, pages 153–162. Springer, 2012.

[173] R. Ugolotti, Y. S. G. Nashed, P. Mesejo, and S. Cagnoni. Algorithm configuration using GPU-based metaheuristics. In *Genetic and Evolutionary Computation Conference Companion*, GECCO'13, pages 221–222, 2013.

[174] R. Ugolotti, Y. S. G. Nashed, P. Mesejo, Š. Ivekovič, L. Mussi, and S. Cagnoni. Particle Swarm Optimization and Differential Evolution for model-based object detection. *Applied Soft Computing*, 13(6):3092–3105, 2013.

[175] J. Vazquez, F. Valdez, and P. Melin. Comparative study of social network structures in PSO. In O. Castillo, P. Melin, W. Pedrycz, and J. Kacprzyk, editors, *Recent Advances on Hybrid Approaches for Designing Intelligent Systems*, volume 547 of *Studies in Computational Intelligence*, pages 239–254. Springer, 2014.

[176] G. Venter and J. Sobieszczanski-Sobieski. Parallel Particle Swarm Optimization algorithm accelerated by asynchronous evaluations. *Journal of Aerospace Computing, Information, and Communication*, 3(3):123–137, 2006.

[177] J. Vesterstrom and R. Thomsen. A comparative study of Differential Evolution, Particle Swarm Optimization, and Evolutionary Algorithms on numerical benchmark problems. In *IEEE Congress on Evolutionary Computation*, CEC'04, pages 1980–1987, 2004.

[178] P. Vidal and E. Alba. A multi-GPU implementation of a Cellular Genetic Algorithm. In *IEEE Congress on Evolutionary Computation*, CEC'10, pages 1–7, 2010.

[179] F. Wang, X. shi He, L. Luo, and Y. Wang. Hybrid optimization algorithm of PSO and Cuckoo Search. In *Conference on Artificial Intelligence, Management Science and Electronic Commerce*, AIMSEC, pages 1172–1175, 2011.

[180] Y. Wang, Z. Cai, and Q. Zhang. Differential Evolution With Composite Trial Vector Generation Strategies and Control Parameters. *IEEE Transactions on Evolutionary Computation*, 15(1):55–66, 2011.

[181] D. Weyland. A Rigorous Analysis of the Harmony Search Algorithm: How the Research Community Can Be Misled by a "Novel" Methodology. *Int. J. Appl. Metaheuristic Comput.*, 1(2):50–60, 2010.

[182] F. Wilcoxon. Individual comparisons by ranking methods. *Biometrics bulletin*, 1(6):80–83, 1945.

[183] M.-L. Wong and T.-T. Wong. Parallel Hybrid Genetic Algorithms on Consumer-Level Graphics Hardware. In *IEEE Congress on Evolutionary Computation*, CEC'06, pages 2973–2980, 2006.

[184] J. R. Woodward and J. Swan. Automatically designing selection heuristics. In *Genetic and Evolutionary Computation Conference Companion*, GECCO'11, pages 583–590, 2011.

[185] J. R. Woodward and J. Swan. The Automatic Generation of Mutation Operators for Genetic Algorithms. In *Genetic and Evolutionary Computation Conference Companion*, GECCO'12, pages 67–74, 2012.

[186] L. Xu, F. Hutter, H. H. Hoos, and K. Leyton-Brown. SATzilla: Portfolio-based Algorithm Selection for SAT. *Journal of Artificial Intelligence Research*, 32(1):565–606, 2008.

[187] F. Xue, A. C. Sanderson, and R. J. Graves. Pareto-based multi-objective Differential Evolution. In *IEEE Congress on Evolutionary Computation*, volume 2 of *CEC'03*, pages 862–869, 2003.

[188] X.-S. Yang, Z. Cui, R. Xiao, A. H. Gandomi, and M. Karamanoglu. *Swarm Intelligence and Bio-Inspired Computation: Theory and Applications*. Elsevier Science Publishers B. V., 1st edition, 2013.

[189] Z. Yang, X. Yao, and J. He. Making a Difference to Differential Evolution. In P. Siarry and Z. Michalewicz, editors, *Advances in Metaheuristics for Hard Optimization*, Natural Computing Series, pages 397–414. Springer, 2008.

[190] M. Ye, X. Wang, R. Yang, L. Ren, and M. Pollefeys. Accurate 3D pose estimation from a single depth image. In *IEEE International Conference on Computer Vision*, ICCV'11, pages 731 –738, 2011.

[191] Q. Yu, C. Chen, and Z. Pan. Parallel Genetic Algorithms on Programmable Graphics Hardware. In L. Wang, K. Chen, and Y. Ong, editors, *Advances in Natural Computation*, volume 3612 of *Lecture Notes in Computer Science*, pages 1051–1059. Springer, 2005.

[192] L. A. Zadeh. Fuzzy Logic, Neural Networks, and Soft Computing. *Communications of the ACM*, 37(3):77–84, 1994.

[193] S. Zhang and Z. He. Implementation of Parallel Genetic Algorithm Based on CUDA. In Z. Cai, Z. Li, Z. Kang, and Y. Liu, editors, *Advances in Computation*

and Intelligence, volume 5821 of *Lecture Notes in Computer Science*, pages 24–30. Springer, 2009.

[194] W.-J. Zhang and X.-F. Xie. DEPSO: hybrid particle swarm with differential evolution operator. In *IEEE International Conference on Systems, Man and Cybernetics*, volume 4, pages 3816–3821 vol.4, 2003.

[195] S.-Z. Zhao, J. Liang, P. Suganthan, and M. Tasgetiren. Dynamic multi-swarm Particle Swarm Optimizer with local search for Large Scale Global Optimization. In *IEEE Congress on Evolutionary Computation*, CEC'08, pages 3845–3852, 2008.

[196] Y.-J. Zheng, S.-Y. Chen, Y. Lin, and W.-L. Wang. Bio-inspired optimization of sustainable energy systems: a review. *Mathematical Problems in Engineering*, 2013.

[197] A. Zhigljavsky and A. Zilinskas. *Stochastic global optimization*. Springer, 2007.

[198] A. Zhou, B. Y. Qu, H. Li, S. Z. Zhao, P. N. Suganthan, and Q. Zhang. Multiobjective Evolutionary Algorithms: A survey of the state of the art. *Swarm and Evolutionary Computation*, 1(1):32 – 49, 2011.

[199] W. Zhu. Massively parallel Differential Evolution–pattern search optimization with graphics hardware acceleration: an investigation on bound constrained optimization problems. *Journal of Global Optimization*, 50(3):417–437, 2011.

Ringraziamenti

Just remember that the last laugh is on you.

– Eric Idle

La prima persona che sento il dovere di ringraziare è il mio relatore Stefano Cagnoni per il suo prezioso supporto, i suoi suggerimenti, il suo costante approccio propositivo e la pazienza di seguire il mio ondivago percorso di questi anni, nonché per l'avermi sempre consentito di lavorare nelle migliori condizioni possibili.

Un doveroso ringraziamento e un abbraccio anche alle persone con cui ho condiviso il laboratorio nel corso di questi anni: Pablo Mesejo, Youssef Nashed e Hamid Hassannejad. La vostra collaborazione, i vari scambi di idee e opinioni, i consigli e suggerimenti sia in ambito lavorativo che al di fuori di esso sono stati fondamentali. E, spesso, molto divertenti.

Un grazie enorme anche a Henesis, in particolare all'"AI team" capitanato da Federico Sassi, per lo sforzo che ha sempre fatto per venire incontro alle mie esigenze e che ha sempre fornito un enorme quantitativo di idee (e cibo), consentendomi di migliorare in tanti ambiti diversi. Nei prossimi anni ci divertiremo...

L'ultimo e il più grande ringraziamento va alla mia famiglia per l'educazione che mi ha dato e per l'incondizionato e immeritato sostegno che mi ha sempre fornito, nonostante tutto.

***

Pablo Mesejo and Youssef Nashed helped in designing the experiments and writing the code of the experiments in Chapter 3. Youssef Nashed is also the co-author of *libcudaoptimize*.

The work in sections 5.2 and 5.3 has been inspired by Luca Mussi's work and has been done in collaboration with Youssef Nashed; the work in section 5.4 has been made in collaboration with Pablo Mesejo under the supervision of Ferdinando Di Cunto and has been funded by Compagnia di San Paolo; the work in section 5.5 has been made in collaboration with Giorgio Micconi.

My PhD programme has been funded by Compagnia di San Paolo and Fondazione Cariparma.