Università di Pisa

Dipartimento di Informatica
Scuola di Dottorato "Galileo Galilei"

Dottorato di Ricerca in Informatica

Ph.D. Thesis

Niching in Particle Swarm Optimization

Alessandro Passaro

Supervisor

Antonina Starita

October 15^{th}, 2007

Abstract

The Particle Swarm Optimization (PSO) algorithm, like many optimization algorithms, is designed to find a single optimal solution. When dealing with multimodal functions, it needs some modifications to be able to locate multiple optima. In a parallel with Evolutionary Computation algorithms, these modifications can be grouped in the framework of Niching.

In this thesis, we present a new approach to niching in PSO that is based on clustering particles to identify niches. The neighborhood structure, on which particles rely for communication, is exploited together with the niche information to perform parallel searches to locate multiple optima. The clustering approach was implemented in the k-means based PSO (kPSO), which employs the standard k-means clustering algorithm. We follow the development of kPSO, starting from a first, simple implementation, and then introducing several improvements, such as a mechanism to adaptively identify the number of clusters.

The final kPSO algorithm proves to be a competitive solution when compared with other existing algorithms, since it shows better performance on most multimodal functions in a commonly used benchmark set.

Contents

Chapter 1

Introduction

The observation of natural phenomena has inspired important and promising approaches in Computer Science. Many heuristic search and optimization techniques have drawn inspiration from physics (simulated annealing [KGV83]), evolution (genetic algorithms [Gol89], genetic programming [Koz92]), neurology (artificial neural networks [Hay99, Bis95, Koh01]), immunology (artificial immune systems [dCT02]), social behavior (ant colony optimization [BDT99], particle swarms [KE01]), and other domains.

In this thesis, we will focus our attention on an important phenomenon observed in nature: social behavior. Many animal species in nature show interesting social interactions, like knowledge sharing or behavioral imitation, with varying degrees of importance and complexity. In general, however, social animals gain a survival advantage by joining in groups and interacting with each other. For instance, schools of fish have an advantage in escaping predators, as each individual fish can be a kind of lookout for the whole group. Herding animals are advantaged in finding food, since if one animal finds a source of food, the others will watch and follow. Other examples we will discuss later are flocks of birds and insect colonies.

The interesting aspect of this kind of processes is that they show *self-organization* and *emergent* characteristics. In fact, social animals behave in a much more complex and adaptable way when they are together than when they are considered alone. Indeed, this is a much desired property for a *computational* model. The possibility of a system, whose single parts are quite simple to program, but whose overall behavior is indeed complex and adaptable, is very appealing. The field of research focused on developing such systems and applying them to solve a wide range of problems is often referred to with the term "Swarm Intelligence".

Ant-colony systems are probably one of the most popular examples of swarm intelligence [BDT99]. They are inspired by the behavior of ants in search for food: when ant colonies are given access to a food source that has multiple approach paths, most ants end up using the shortest and most efficient route. To expedite this process, the ants deposit a chemical substance, called pheromone, on the ground when traveling from the nest to the food source. While the process iterates, pheromone is deposited at a higher rate on the shorter paths than on the longer ones. When other ants arrive at a decision point, such as an intersection between various paths, they tend to follow the path along which they smell the highest amount of pheromone. After several trips, nearly all of the ants end up using the shortest path, due to the high concentration of pheromone deposited.

In computer programs, ant-colony algorithms simulate basically the same process, where paths are candidate solutions to the problem at hand [DC99]. The optimal path, that is the best solution, is thus identified by the highest concentration of virtual ants traveling on it. Ant-colony algorithms have been used to solve numerous optimization problems, ranging from the classical traveling salesman problem [DG97] to routing in telecommunication networks [GSB02].

Another promising technique in the field of swarm intelligence, on which we will focus in this thesis, is the Particle Swarm Optimization (PSO) algorithm, inspired by social behavior of bird flocking, fish schooling or bugs swarming.

In a particle swarm, the potential solutions are particles which fly through the problem space by following the current best particles of the swarm. Each particle keeps track of the coordinates in the problem space associated with the best solution it has achieved so far. Another *best* value tracked by the particle swarm is the global one, that is the best value obtained so far by any particle in the swarm. At each time step, particles change their velocities, accelerating towards previous best locations. As a whole, the swarm moves on the problem space with peculiar dynamics, which provide for an interesting search methodology.

In the past years, the particle swarm approach has been successfully applied in many research and application areas, proving to be a powerful and effective optimization algorithm. Among the application areas in which PSO has been used are classification, pattern recognition, data mining [SSN03], clustering [XDE+03], training of Support Vector Machines [PE03], biological system modeling, protein motif discovery [CRHW04], scheduling (planning), signal processing, games [FE03], robotic applications, system design and control [YKFN99]. Moreover, it has been used in conjunction with other methodologies, such as Artificial Neural Networks

(ANN), showing great potential. In particular, the use of PSO to replace the back-propagation learning algorithm in ANNs has been proposed since its first conception and in the last years many researches confirmed its validity [ESD96, ES98b, EH99, KE01, GV03].

This thesis originates from a study on the adaptation of the particle swarm approach to a particular field, that of multimodal function optimization. In fact, the design of standard optimization algorithms, and of the PSO algorithm among them, is usually targeted on the goal of finding a single optimal solution for a given problem. However, in many situations setting this kind of goal is not a good choice. Some problems have multiple possible solutions which are optimal according to the criterion which has been chosen to drive the optimization. In such a case, an optimization algorithm should ideally find all the optimal solutions. Then, these solutions can be used in many different ways, which strongly depend on the specific application field. In some cases, a successive selection process can be employed to choose a particular solution among those retrieved by the optimization algorithm, but using different criteria. Another possibility is to maintain a set of different solutions and use some or all of them in different situations. In certain applications, it is even possible to combine multiple solutions to a problem to build a new, better one.

A formal and general way to treat the study of optimization problems is by identifying the objective function which will be used to evaluate candidate solutions. When such a function presents multiple global optima, or local optima whose values are very close to the global one, the problem, and the function itself, are said to be *multimodal*. In dealing with multimodal functions, the behavior of a standard optimization algorithm may be less then ideal. In some cases, in fact, an algorithm designed to look for a single optimum would arbitrarily pick just one of the optimal solutions. However, often it would even be misled by the presence of more than a single optimum and fail to reach a valid solution.

In general, it is possible to follow two different approaches to develop an optimization algorithm which can deal with multimodal functions. The first is to design a specific algorithm from the start, the second is to modify an existing one in order to adapt it for this kind of problems. The latter approach has been taken, for example, in the field of Evolutionary Computation (EC). In particular, the optimization algorithms based on the principles of darwinian evolution have been modified by introducing the concept of *niching*. This concept is based on the observation that in an environment with limited resources, the evolutionary

process leads to the emergence of different species, which tend to exploit different environmental niches, specializing in different tasks.

Niching techniques are an attempt to reproduce these phenomena in search and optimization algorithms based on EC. Such techniques allow the search algorithm to divide the problem space into different areas, which correspond to the multiple optimal solutions, and to evolve different solutions, or *species*, in each of them. In this way, evolutionary algorithms with niching can effectively be applied to multimodal optimization problems.

Although the particle swarm is based on a different paradigm than evolutionary algorithms, the two approaches have enough in common to allow the application of niching techniques also in PSO. Both approaches, in fact, proceed by simulating a population of individuals which undergo some kind of transformation over time. Even if this transformation is different in the two cases, being an evolutionary process in one, and a flight over the function landscape in the other, it is nonetheless based on some kind of interactions between the elements of the populations. Moreover, both approaches have a strong stochastic component which is crucial in providing their typical robustness.

When introducing niching methodologies in the particle swarm, however, it is important to consider also the fundamental differences it has from the EC algorithms. Specifically, an aspect in which the two approaches really differ, and which is of crucial importance in the application of niching techniques, is the way the elements in the population interact with each other. While in EC, the interaction is rather indirect, as it is mediated by the selection process, in PSO it is much more direct and involves a sort of exchange of knowledge between particles. In particular, it is tied to the specific social structure in which the swarm is organized, and that is commonly called the *neighborhood topology* of the swarm. To obtain the formation of multiple niches, any technique would have to actively intervene, in a way or in another, on the swarm's neighborhood topology, and thus alter the standard interactions among particles.

Another peculiarity of the PSO algorithm lies in the memory particles keep of the previous positions they have visited. This memory, which has no correspondent in the evolutionary framework, plays a critical role in the swarm's dynamics, and can also be useful in designing a niching approach.

In our work on niching for the PSO algorithm, we exploited these peculiarities of the particle swarm, together with another of its characteristics. Particles in a swarm, in fact, naturally tend to group around regions of the search space which are

close to optimal solutions. In the classical swarm, this tendency is then balanced by the attraction towards the best solution the swarm has visited, where all the particles converge in the end of a run.

Our approach to niching is founded on the idea of identifying these natural groups during the run using a clustering algorithm. Specifically, instead of using the current positions of particles, the algorithm performs a clustering based on the best positions they keep in their internal memory. In this way, the particles are grouped according to the position towards which they are actually attracted. The clusters obtained are then naturally exploited to delimitate niches in the population. To this end, the neighborhood topology, which in general in the classical PSO is fixed at the beginning of the simulation, is in this case adapted during the run. Particles are constrained to communicate only with other particles in the same cluster (niche). In this way, the swarm will be divided in several sub-swarms which will perform each a local search for a specific solution.

In the following, we will introduce a first implementation of this approach, which uses the k-means algorithm, one of the most popular clustering algorithms. Our niching PSO algorithm has thus been called k-means Particle Swarm Optimization (kPSO). The new algorithm will be analyzed in details and improved with a series of enhancements to its first implementation. The most important goals of the refining process which will lead to the final version of the kPSO algorithm are:

- Obtaining an optimization algorithm that is enough flexible so that it can be applied to a wide variety of different problems without requiring too much parameter tuning. Specifically, in our case, this means that we should contain the number of new parameters introduced by adding the niching approach to the standard PSO algorithm, or at least that the new parameters should not be critically dependent on the problem at study.

- Demonstrating that the final version of the kPSO algorithm can be compared and shows some advantages over the most effective among the other niching approaches based on Particle Swarm. The comparison should take into account primarily the efficiency in terms of number of function evaluations required to locate all the optima of the benchmark functions, but also the computational overhead added with the niching procedure.

The discussion of our research is organized as follows. This thesis starts with an introduction to the Particle Swarm Optimization algorithm, presented in

Chapter 2. In the same chapter, we will also examine the relation between PSO and Evolutionary Computation algorithms, and analyze the important role of the neighborhood structure of the swarm.

In Chapter 3 we will discuss the second main topic of this thesis, which is the optimization of multimodal functions. In particular, we will focus on the introduction of niching approaches in the field of EC algorithms. Then, we will review existing applications of niching techniques in the PSO field.

Our clustering approach to niching will be presented in Chapter 4. Here we will discuss the concept of clustering applied to the particles of the swarm and the specific algorithm we chose to employ, the k-means clustering algorithm. We will introduce our first implementation of the clustering approach, the kPSO algorithm. Then, we will follow in details the development of the algorithm, by testing it on different functions and comparing it with other approaches at each step. At the end of the chapter, the final version of the algorithm will be presented, which uses an approach based on the Bayesian Information Criterion to adaptively determine the number of clusters during the simulation. The performance of this algorithm will thus be compared to that of other significant PSO niching algorithms in the final set of experiments.

Finally, in Chapter 5, we provide a summary of the work that has been carried out, in which we examine the current results and try to delineate future research directions.

Chapter 2

Particle Swarm Optimization

Particle Swarm Optimization (PSO) is a relatively new family of algorithms which can be used to find optimal (or near optimal) solutions to numerical and combinatorial problems. It is easily implemented (the core of the algorithm can be written in a few lines of code) and has proven both very effective and quick when applied to a diverse set of optimization problems.

PSO was originally developed by Kennedy and Eberhart in 1995 [KE95], taking inspiration both from the related field of evolutionary algorithms and in artificial life methodologies. In fact, in the next sections we will discuss whether PSO can actually be considered an evolutionary algorithm, pointing out main similarities and differences with respect to (other) evolutionary techniques, in particular genetic algorithms and evolutionary programming. First of all, however, we will briefly depict the origins of the PSO algorithm, which are strongly tied to the artificial life theory of bird flocking, fish schooling and swarming.

2.1 The Origins of the algorithm

2.1.1 Bird flocking simulations

Animal social behavior such as that seen in flocks, schools, or herds, has always attracted many researchers, interested in discovering the underlying rules which enable, as an example, large numbers of birds to flock synchronously, often scattering and regrouping, and suddenly changing direction. Many models of this *flocking behavior* were also used to create computer simulations, such as those by Heppner and Grenander [HG90], and by Reynolds [Rey87].

These simulations relied on models which focused on local processes, thought to underlie the unpredictable group dynamics of the birds in a similar way the simple local rules for cell updating can yield complex global dynamics in cellular automata [vN51, vNB66, Bur70]. In particular, they explained the *flocking behavior* as the overall result of the individual birds' efforts to maintain an optimal distance between themselves and their neighbors.

A simple, although quite realistic, graphical simulation of a flock of birds was realized implementing two rather simple properties: nearest-neighbor velocity matching and *craziness*. A population of birds was represented by a set of points moving on the screen, whose starting positions and velocities were randomly assigned. At each step in the simulation, each bird set its own velocity to match that of its nearest neighbor, thus reproducing the flock's beautiful synchrony of movement. Moreover, a stochastic factor, *craziness*, introduced a slight change in some randomly chosen velocities, giving the simulation a more interesting and almost *life-like* appearance.

Other bird simulations, realized by Heppner, introduced a dynamic force, attracting birds towards a target position on the screen, identified as a *roost* or as a *cornfield*. In this way the *craziness* factor became unnecessary, because the simulation took on a life of its own. In fact,

> [...] the flock swirled around the goal, realistically approaching it, swinging out rhythmically with subgroups synchronized, and finally *landing* on the target. ([KE95])

Such a kind of behavior was obtained adding the following rules:

1. Each bird evaluated its position in term of its distance from the target:

$$E(x, y) = \sqrt{(x - x_t)^2 + (y - y_t)^2} \tag{2.1}$$

   where (x_t, y_t) was the position of the target.

2. When updating its velocity, each bird *remembered* the best position it had previously visited and adjusted its velocity on each axis by a random amount (weighted by the p_{incr} parameter) in that direction. Moreover it *knew* the best position ever found by any member of the flock and performed another velocity adjustment towards it (using another parameter: g_{incr}). Varying the two

increment parameters, one could obtain different flock behaviors, essentially controlling the speed with which the flock approached the target.

The introduction of a *roost* (or *cornfield*) in the second variation of bird simulation led to a lot of intriguing questions. While in the simulation the target was at a fixed known position, in real life birds in a flock can efficiently find food, without previously knowing its location nor appearance. It seems possible that flock's dynamics enable single members to gain advantage of each others' experience. Citing sociobiologist E. O. Wilson, although in reference to fish schooling:

> In theory at least, individual members of the school can profit from the discoveries and previous experience of all members of the school during the search for food. This advantage can become decisive, outweighing the disadvantages of competition for food items, whenever the resource is unpredictably distributed in patches" ([Wil75], p. 209).

Indeed individuals group in herds, flocks, schools or swarms in order to avoid predators, seek food and mates and so on, thus gaining an evolutionary advantage by socially sharing information. The key aspect in this case is the *cooperation* between conspeciates to reach a certain goal. In the previous simulation, for example, a bird which had found a good position (near the target) *led* its neighbors towards it, thus increasing the odds that they would all find the target. This is perfectly in line with the view by which intelligence is actually a social process [KE01, Ken97].

It should be noted that the simulated birds have no direct knowledge of the target position, although they can evaluate their distance to it. In fact they are actually minimizing this distance, as expressed by the evaluation function E in Equation (2.1). What if we replace E with an arbitrary function? We could thus employ our flock of birds to look for the function's minima, transforming an artificial life simulation in a function optimization algorithm.

2.1.2 Function optimization

A general way to search for a solution of a problem is to determine an *objective*, or *cost* function which describes the problem and to optimize it. Thus the field of function optimization is of wide interest. Optimizing a function $f(x)$ means either *minimizing* or *maximizing* it, however one problem can be easily converted into the other by considering the function with the reverse sign:

$$f'(x) = -f(x). \tag{2.2}$$

Thus, although in the rest of this thesis we will face both kind of problems, now we can define the next concepts considering only the problem of minimization, without in fact losing generality. The minimization of a function $f : \Omega \to \mathbb{R}$ can be formally described as finding the *global minimizer* $\mathbf{x}^*$, such as

$$f(\mathbf{x}^*) \leq f(\mathbf{x}),\ \forall \mathbf{x} \in \Omega. \tag{2.3}$$

In this thesis we will generally consider $\Omega \subseteq \mathbb{R}^d$ and thus $\mathbf{x} = (x_1, \ldots, x_d)$.

Sometimes it is acceptable or it is only possible to find *local minimizers*, that is solutions which minimize the function in a subset of Ω. When considering $\Omega \subseteq \mathbb{R}^d$, a local minimizer $\mathbf{x}^*_\ell$ must satisfy

$$\exists \epsilon > 0 : f(\mathbf{x}^*_\ell) \leq f(\mathbf{x}),\ \forall \mathbf{x} : \|\mathbf{x} - \mathbf{x}^*_\ell\| < \epsilon. \tag{2.4}$$

Thus a local minimizer $\mathbf{x}^*_\ell$ is the point with the lowest function value of its neighborhood, which is the set of points whose distance from $\mathbf{x}^*_\ell$ is less than a certain ϵ.

2.1.3 The optimization algorithm

The particle swarm bears much resemblance to flocks' simulations, but in place of birds we have, more generically, *particles* flying around following similar simple rules. From an optimization point of view, it is straightforward to see the particles' flight as a trajectory in the solution space. In this way the PSO algorithm can be used to minimize a *generic* function in the form: $f(x, y) : \mathbb{R}^2 \to \mathbb{R}$, by simply replacing the function E in Equation (2.1) with f.

By the way, since we are not seeking a graphical simulation, we are no more limited to a two-dimensional (or three-dimensional) space, but we can consider particles moving in $\mathbb{R}^d$ and an evaluation function $f : \mathbb{R}^d \to \mathbb{R}$. Thus from now on we will describe the basic PSO version used to solve the minimization problem of a function in $\mathbb{R}^d$. As a matter of fact, however, different optimization problems can be faced with specific variations in the algorithm, some straightforward, like the switch to a maximization problem, others a little more complex, like the use of a binary encoding [KE97].

Algorithm 2.1 The pseudocode for the Particle Swarm Optimization in its standard version.

```
1: procedure PSO
2:     Initialize particles with random positions and velocities.
3:     Set particles' pbests to their current positions.
4:     Calculate particles' fitness and set gbest.
5:     for T generations do
6:         Update particles' velocities.
7:         Update particles' positions.
8:         Recalculate particles' fitness.
9:         Update particles' pbest and gbest.
10:    end for
11: end procedure
```

For the same reason, the choreographic effect of the flock's flight is no longer needed, so the nearest-neighbor velocity matching can be removed. Although the general effect is more like a *swarm* than like a *flock*, experiments have shown that optimization actually occurs slightly faster in this way.

Finally, another revision to the algorithm involves the way velocities are updated. In previous simulations the velocity along a given axis was incremented or decremented by a random amount if the best known position was respectively after or before the current position along that axis. It seems much simpler, and empirically has proven more performing, to adjust velocities by an amount proportional to the (signed) distance of the best location from the current one (along each axis).

The pseudocode for the resulting PSO algorithm, in its simplest version, is given in Algorithm 2.1. The rules for velocity and position updating used in the algorithm are described in details in the following section.

2.1.4 PSO equations

Let d be the dimension of the search space, then $\mathbf{x}_i = (x_{i1}, x_{i2}, \ldots, x_{id})$ denotes the position of the particle $i \in (1, 2, \ldots, N)$ of the swarm, and $\mathbf{p}_i = (p_{i1}, p_{i2}, \ldots, p_{id})$ denotes the best position it has ever visited. The index of the best particle in the population (the one which has visited the global best position) is represented by the symbol g. At each time step t in the simulation the velocity of the i^{th} particle,

represented as $\mathbf{v}_i = (v_{i1}, v_{i2}, \ldots, v_{id})$, is adjusted along each axis j following the equation:

$$v_{ij}(t+1) = v_{ij}(t) + \varphi_p \cdot (p_{ij}(t) - x_{ij}(t)) + \varphi_g \cdot (p_{gj}(t) - x_{ij}(t)) \tag{2.5}$$

where φ_p is a random number uniformly distributed in $[0, p_{incr}]$ and φ_g is a random number uniformly distributed in $[0, g_{incr}]$. p_{incr} and g_{incr} are the same positive constants used in flock's simulations and are respectively called the *cognitive* and *social acceleration coefficient*.

Moreover, the particle's velocity can be constricted to stay in a fixed range, by defining a maximum velocity value V_{max} and applying the following rule after every velocity updating:

$$v_{ij} \in [-V_{max}, V_{max}] \tag{2.6}$$

In this way the likelihood of particles leaving the search space is reduced, although indirectly, by limiting the maximum distance a particle will cover in a single step, instead of restricting the values of $\mathbf{x}_i$.

The new position of a particle is calculated using:

$$\mathbf{x}_i(t+1) = \mathbf{x}_i(t) + \mathbf{v}_i(t+1). \tag{2.7}$$

The personal best position of each particle is updated using:

$$\mathbf{p}_i(t+1) = \begin{cases} \mathbf{p}_i(t) & \text{if } f(\mathbf{x}_i(t+1)) \geq f(\mathbf{p}_i(t)) \\ \mathbf{x}_i(t+1) & \text{if } f(\mathbf{x}_i(t+1)) < f(\mathbf{p}_i(t)) \end{cases} \tag{2.8}$$

while the global best index is defined as:

$$g = \arg\min_i f(\mathbf{p}_i(t+1)), \qquad 1 \leq i \leq N. \tag{2.9}$$

An essential feature of the PSO algorithm is the way in which the local and global best positions, $\mathbf{p}_i$ and $\mathbf{p}_g$, and their respective acceleration coefficients, are involved in velocity updates. Conceptually, $\mathbf{p}_i$ (also known as *pbest*) resembles the particle's autobiographical memory, i.e. its own previous experience, and the velocity adjustment associated with it is a kind of *simple nostalgia*, as it leads the particle to return in the position where it obtained its best evaluation. On the other hand, $\mathbf{p}_g$ (*gbest*) is a sort of group knowledge, a common standard which every single particle seeks to attain.

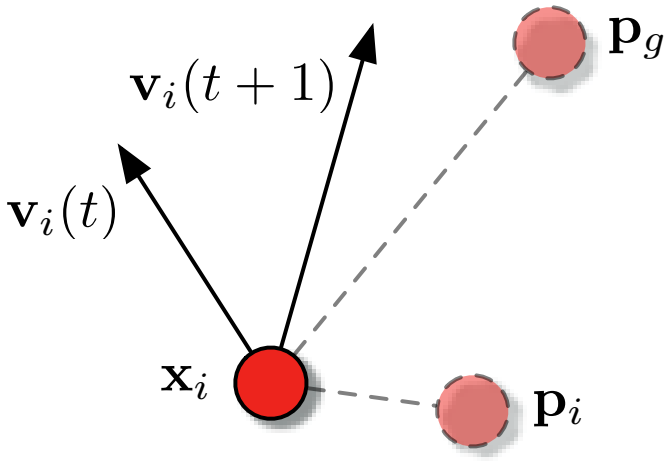


Figure 2.1: At each step t a particle i updates its velocity and position. The new velocity $\mathbf{v}_i(t+1)$ is the sum of three terms: the previous velocity $\mathbf{v}_i(t)$, and two terms proportional to the distance from $\mathbf{p}_i$, the best position visited so far by the particle, and from $\mathbf{p}_g$, the best position visited so far by the whole swarm. The new position of the particle is then computed by just adding the new velocity.

The overall effect is such that when particles find a good position, they begin to look nearby for even better solutions, but, on the other hand, they continue to explore a wider area, likely avoiding premature convergence on local optima and realizing a good balance between exploration of the whole search space and exploitation of known good areas [OM98].

2.1.5 The particle swarm

While in the previous paragraphs we have discussed quite in details the origins of the PSO algorithm, it may still be unclear where its name, "Particle Swarm", exactly comes from.

In their original paper [KE95], Kennedy and Eberhart discuss their choice of the term *particle* to denote the members of the swarm, as mainly due to their physical behavior: they have velocities and accelerations, although can be considered to have arbitrarily small masses and volumes. Moreover they refer to the models developed by Reeves [Ree83] for diffuse objects such as clouds, fire, and smoke, constituted by *particle systems* which were based on quite similar rules for the simulation of particles' movements.

The term *swarm* was used as to better describe the behavior of the particles,

since, as discussed above, along the path from a flock simulation to an optimization algorithm, they lost their choreographic movements. However, the term has also references in artificial life literature. In particular Millonas [Mil94] introduced five basic principles to define what he called *swarm intelligence*. This principles can easily be seen to apply in the particle swarm:

1. **Proximity** – particles can perform simple space and time computations as they fly through a d-dimensional space;
2. **Quality** – particles can respond to environmental quality factors as they are accelerated towards *pbest* and *gbest*;
3. **Diverse Response** – the population as a whole does not commit its activities along excessively narrow channels: this is ensured by the competition between *pbest* and *gbest*;
4. **Stability** – the population mode of behavior does not change with every change in the environment, since it does not change until *gbest* changes;
5. **Adaptability** – the population can change its behavior according to the environment, when it's worth the computational price: this is actually the reverse coin of the stability principle; the particle swarm adheres to it since it *does* change when *gbest* changes.

2.2 Comparison with evolutionary computation

The particle swarm has so much in common with the various Evolutionary Computation (EC) paradigms that some authors claim it could be considered itself a kind of evolutionary algorithm. Even without going so far, however, there is no doubt that it shares many features and many goals with EC methods [ES98a, Ang98a]. First of all, they take inspiration by two natural processes, namely evolution of species and social interaction, which are strongly related. In fact, many species of animals put in effect social strategies which give them an evolutionary advantage. Besides, an intriguing, although much discussed, view of cultural evolution, proposed by Dawkins [Daw76], points out the similarity between cultural transmission in social processes and genetic transmission in natural evolution.

Even from an implementation point of view, the two methods show striking similarities: they both maintain, and adapt throughout time, a population of

individuals, which represent problem's candidate solutions. The adaptation process is led by a *fitness* measure which explicitly refers to the environment adaptation in EC, but is substantially identical to the evaluation measure in PSO. Moreover, it proceeds by modifying population members both by a random factor and by considering the interactions between them.

In particular PSO can be considered to lie somewhere between Genetic Algorithms (GA) and Evolutionary Programming (EP). Like the latter, in fact, it is highly dependent on stochastic processes. However, the adjustment towards *gbest* and *pbest* in PSO is conceptually similar to the crossover operator in GA.

2.2.1 Competition vs cooperation

The particle swarm model, however, differs from the evolutionary approach in some relevant aspects. The main difference is connected with one of the foundations of evolutionary theory: the *survival of the fittest*. In fact, in EC methods, at each iteration (or *generation*) population members are selected in function of their *fitness*. The best individuals are reproduced in the next generation, while less fit individuals are likely to be deleted. Thus the population as a whole evolves by constantly letting its members *compete* against each other, replacing its members and promoting the fittest ones.

Instead in the particle swarm, population members are never replaced. The initial number of particles remains constant throughout the whole simulation. Rather, single particles move through the space, changing their position and velocity, thus individually *evolving* through time. Moreover, instead of fighting against each other for the chance to survive and/or reproduce, they directly interact with each others, exchanging their knowledge about function's landscape, thus *cooperating* to reach optima.

2.2.2 Particles' trajectories

Another major difference between particle swarms and EC methods is that particles' *velocities* are adjusted, rather than their *positions*. In EC a new candidate solution is generated – typically by crossover or mutation – acting upon its position in the search space, as well as its parents' ones. Instead, in PSO both the interaction with other particles and the stochastic factor influence the velocity of a particle. This is responsible for the characteristic trajectories of the particles in a swarm, and results

in a quite effective search.

In fact, much of the success of particle swarms seems to lie in particles' tendency to hurtle past their target. Holland's chapter on the "optimum allocation of trials" [Hol92] points out the delicate balance between conservative testing of known regions versus risky exploration of the unknown. In PSO, the stochastic factors allow a thorough search of areas which are close to relatively good regions. Moreover, the momentum effect, caused by the velocity update formula which modifies the current velocities rather than replacing them, results in overshooting, thus in exploration of unknown regions of the problem domain.

2.3 Enhancements

As an optimization algorithm, the particle swarm is an attractive choice and has a number of desirable properties, including simplicity of implementation and good empirical performance, even compared to evolutionary algorithms such as GAs.

Even so, it is not without problems: the basic PSO can suffer from premature convergence, tending to get stuck in local minima. It can also suffer from an ineffective exploration strategy around local minima, and thus in some cases it does not find good solutions as quickly as it could. Moreover, like for most heuristic algorithms, there is also the problem of adjusting the tunable parameters to obtain improved performances.

In the following we will briefly review some of the many enhancements to the basic PSO algorithm which have been proposed to address these issues.

2.3.1 Parameters selection

A common problem of many biologically-inspired algorithms is the high number of parameters they need and, even worse, their dependence upon the problem to solve. The basic version of the particle swarm we have discussed so far requires only a few parameters, as can be seen in the equations reported in Section 2.1.4. Moreover the original authors suggested a set of values by which the algorithm actually worked quite well in various experiments, including optimization of many test functions and neural networks weights training. The four parameters to set are: N, the number of particles, V_{max}, the maximum allowed velocity, p_{incr} and g_{incr}, the two acceleration coefficients.

The experiments conducted by the original authors on a series of test functions with low dimensionality, had all $N = 20 \div 60$ and subsequent literature on the subject essentially agrees on this order of magnitude [SG05]. In fact, such a low number of individuals is quite unusual in population-based methods, even for quite simple test functions. In the field of genetic algorithms, for example, populations with hundred of members are more than common. For the PSO to require a small number of particles is a real advantage, since the computational resources employed are drastically reduced.

The V_{max} parameter sets the maximum velocity particles are allowed to gain at each iteration. In fact, the two random weighted terms in Equation (2.5) can lead the swarm to a kind of explosion or *drunkard's walk* [OM99, CK02], as particles velocities and positions tend to infinity. The V_{max} function is to contain this explosion, reducing the likelihood of particles wandering far away in the search space. Moreover, it determines the resolution of the search near best-so-far solutions: with a high value, particles might fly past good solutions, with a too low one, they may not explore enough and become trapped in local optima. In early experiments V_{max} was empirically set to about $10 \div 20\%$ of the dynamic range on each dimension.

The *cognitive* (p_{incr}) and *social* (g_{incr}) acceleration coefficients determine the intensity of particles attraction towards respectively *pbest* and *gbest*. A high value of the *cognitive* coefficient, with respect to the *social* one, results in excessive wandering of single particles through the problem space, while a relatively high *social* coefficient results in the swarm rushing prematurely towards local minima. Instead, approximately equal values of the two acceleration coefficients seem to result in the most effective search of the problem domain, realizing a dynamic balance between the attraction towards single particles' best-so-far position and the attraction towards the global one. Usually in experiments these two parameters are both set to $p_{incr} = g_{incr} \simeq 2$, as to ensure half the probability of flying past the current best position, thus exploring the nearby space.

In the end, the basic PSO was applied essentially unchanged on all test problems, with the only exception of the V_{max} parameters, which still required manual tuning. In order to eliminate the need for such a tuning, many modifications to the basic algorithm have been proposed. The two most important modifications that we are going to discuss involve respectively the introduction of an *inertia weight* and of a *constriction factor*.

Further studies on parameters selection and its influence on particles dynamics can be found in [EGEHSK02, ZMZQ03, RHW04, Tre03, BPV02, CD01].

Inertia weight

The concept of an *inertia weight* was introduced by Shi and Eberhart in [SE98a, SE98b] in order to better control the balance between exploration and exploitation, and to eliminate the need to manually set the V_{max} parameter. The velocity-updating formula in Equation (2.5) is replaced with the one in Equation (2.10), where the inertia weight w is included as a multiplying factor for v_{ij}:

$$v_{ij}(t+1) = w \cdot v_{ij}(t) + \varphi_p \cdot (p_{ij}(t) - x_{ij}(t)) + \varphi_g \cdot (p_{gj}(t) - x_{ij}(t)). \tag{2.10}$$

The inertia weight modifies the influence the particle's previous velocity bears on the new one: as an extreme, setting it to zero would result in no influence at all (so no *inertia*). Suitable selection of the inertia weight provides a balance between global and local exploration and exploitation, and results in fewer iterations on average to find a sufficiently optimal solution. As originally developed, w was often linearly decreased during a run from about 0.9 to 0.4. This resulted in a sort of *cooling off* of the system, as particles began to slow down towards the end of a run, and to better perform a local search near the best-so-far solution.

Although the introduction of the inertia weight made unnecessary the constriction operated by V_{max}, many experiments showed that including a velocity clamping with V_{max} set to the dynamic range of the variables resulted in better performances.

In fact, the inertia weight introduction replaces the choice of the value for V_{max}, with the one for w. However, the latter seems to be much more problem-independent, since the one suggested above performed very well in a number of applications.

Constriction factor

Another method for controlling the behavior of the particle swarm is the introduction of a *constriction factor*. Such a method was first proposed by Clerc in [CK02], in the framework of his studies of the mathematical foundations of the particles' dynamics. In its simplest form – Clerc's Type 1" – the constriction factor is a coefficient χ applied to both terms of the velocity updating formula:

$$v_{ij}(t+1) = \chi \cdot (v_{ij}(t) + \varphi_p \cdot (p_{ij}(t) - x_{ij}(t)) + \varphi_g \cdot (p_{gj}(t) - x_{ij}(t))). \tag{2.11}$$

$$\chi = \frac{2}{|2 - \varphi - \sqrt{\varphi^2 - 4\varphi}|}, \qquad \text{where } \varphi = p_{incr} + g_{incr} > 4. \tag{2.12}$$

Typically, when Clerc's constriction method is used, φ is set to 4.1, thus $\chi \approx 0.73$, simultaneously damping the previous velocity term and the two acceleration terms.

This constriction method results in particles' convergence over time; that is, the amplitude of the individual particle's oscillations slowly decreases as it focuses on a previous best point. Although in this way particles converge to a point over time, another factor in the paradigm prevents the collapse of their trajectories, that is the fact that the target point is actually a stochastically weighted average of two points, *pbest* and *gbest*. In fact, if the swarm is still exploring various optima, and a particle's own previous best is in a different region from the global one, then the particle's cycles will remain wide. As particles begin to cluster in the same optimal region, their trajectories will become narrower, intensely exploiting a focused region of the search space.

However, if a new optimum is discovered, trajectories are free to expand again, thus switching from exploitation to exploration. In this regard the constriction method seems better than the inertia weight with its irreversible cooling off mechanism.

2.3.2 Hybridization with EC operators

An interesting line of research is aimed at introducing features from the Evolutionary Computation methods into Particle Swarm Optimization. The idea is to develop hybrid systems which can exploit the strengths of both paradigms.

Selection

One of the first attempts was the introduction of a *selection* mechanism by Angeline [Ang98b]. The main point is to let the swarm focus its resources in promising areas, replacing less performing particles with new ones placed near the global best. Thus, a tournament selection method is incorporated in the particle swarm and applied on each iteration before normal velocity updates. The selection operator allows to rank particles according to the number of particles they beat in the tournament. Then all the particles in the lower half of the ranked swarm have their positions replaced with those of the particles in the upper half, whilst their

personal best positions remain unchanged. The resulting hybrid system showed some advantages over the standard one, although only on a limited set of functions.

Breeding

Løvbjerg *et al* applied a *breeding* operator [LRK01] which gives particles the ability to generate offsprings. The method works by randomly selecting a subset of particles with a uniform breeding probability p_b, thus independently of their fitness. The breeding particles are then randomly paired off until the breeding set is empty. A crossover operator is applied to the parent particles i and j to generate their offsprings. The offspring positions are calculated by:

$$\mathbf{x}'_i = p_i\mathbf{x}_i + (1-p_i)\mathbf{x}_j \tag{2.13}$$

$$\mathbf{x}'_j = p_i\mathbf{x}_j + (1-p_i)\mathbf{x}_i \tag{2.14}$$

where p_i is from a uniform distribution on $[0,1]$. The new velocities are set to the normalized sum of the parents' velocities:

$$\mathbf{v}'_i = \frac{\|\mathbf{v}_i\|}{\|\mathbf{v}_i+\mathbf{v}_j\|}(\mathbf{v}_i+\mathbf{v}_j) \tag{2.15}$$

$$\mathbf{v}'_j = \frac{\|\mathbf{v}_j\|}{\|\mathbf{v}_i+\mathbf{v}_j\|}(\mathbf{v}_i+\mathbf{v}_j). \tag{2.16}$$

The values for the personal best positions is inherited from the parents particles. The introduction of the breeding operator improves the diversity of the swarm population, and may result in avoiding stagnation on local optima.

The LifeCycle model

Other approaches try to integrate the evolutionary and the swarm methods in different ways. As an example, inspired by the idea of life cycle stages found in nature, the LifeCycle model [KL02] proposes to use a population of candidate solutions which can modify their behavior during the simulation. In fact, each member of the population, depending on its recent search progress, can decide to behave as a particle of a swarm, as an individual of a genetic algorithm, or as a stochastic hill climber. In this way, the optimizer can adapt to the characteristics of the given function, using the specific *mode of behavior* which performs the most successful search.

2.3.3 Other enhancements

Division of Labour

The *division of labour* PSO (DoLPSO) modifies the standard PSO algorithm by introducing a mechanism which improves convergence around optima [VRK02]. In the DoLPSO particles can alternatively perform two tasks:

Task 1: Exploring the search space using the standard PSO.

Task 2: Performing a *local search*: the particle is moved to the global best position found (*gbest*) and its velocity is randomly reinitialized to be no larger than that of the global best particle.

The concept of division of labour involves a task switching process, which constantly monitors each particle and tends to reassign it to the alternative task with a higher probability the higher number of iterations it passed without improving its *pbest*. Particles switching to the local search task decrease swarm diversity and encourage a faster and improved convergence towards local optima.

The GCPSO

In-depth analysis of the PSO algorithm with inertia weight (but the same apply for the constriction factor version) shows how it does not guarantee convergence to either a global or local best solution, but only to the best position found by the swarm [vdB02]. A possible cause is the behavior of particles around the *gbest* position: in particular, when for a particle i it happens that $\mathbf{x}_i = \mathbf{p}_i = \mathbf{p}_g$, the velocity update in Equation (2.10) depends only on the previous velocity term $w\mathbf{v}_i(t)$. Thus when a particle approaches *gbest*, its velocity will tend to zero, which causes the swarm to eventually converge on that position, preventing exploration of other areas.

Van den Bergh and Engelbrecht introduced a new algorithm, the Guaranteed Convergence Particle Swarm Optimizer (GCPSO), which actively counters this behavior and ensure convergence to a local optimum [vdBE02]. The GCPSO works by modifying the rule to update the velocity of the best particle of the swarm:

$$v_{gj}(t+1) = -x_{gj}(t) + p_{gj}(t) + w \cdot v_{gj}(t) + \rho(t) r_j \tag{2.17}$$

where r_j is a sequence of uniform random numbers sampled in $[-1, 1]$. $\rho(t)$ is a scaling factor determined using:

$$\begin{aligned} \rho(0) &= 1.0 \\ \rho(t+1) &= \begin{cases} 2\rho(t) & \text{if } \#successes > s_c \\ 0.5\rho(t) & \text{if } \#failures > f_c \\ \rho(t) & \text{otherwise} \end{cases} \end{aligned} \tag{2.18}$$

where s_c and f_c, are tunable threshold parameters. Whenever the best particle improves its personal best position, the successes counter is incremented and the failures counter is set to 0 and vice versa. The successes and failure counters are both set to 0 whenever the best particle changes. Note that only the best particle of the swarm uses the modified velocity update in Equation (2.17); the rest of the swarm uses the normal inertia weight update rule in Equation (2.10).

The modifications introduced with GCPSO cause the best particle to perform a directed random search around its best position in the search space. This random search can dramatically speed up movement towards an optimum and also guarantees the convergence to at least a local optimum [vdB02].

Self-Organized Criticality

Løvbjerg and Krink introduced an enhancement to the particle swarm applying the principles of the Self-Oganized Criticality (SOC) model [LK02]. The SOC theory was introduced in 1987 by Per Bak, Chao Tang, and Kurt Wiesenfeld to explain common characteristics of different complex systems which are in a state at the border of stability and chaos [BTW87]. SOC is based on the observation of the effects that local interactions between many components in an open system can have on a global scale. In fact, most state transitions between the components only affect their neighborhood, but once in a while entire avalanches of propagating state transitions can lead to a major reconfiguration of the system [Bak96]. Some principles of the SOC theory had been previously applied also in an evolutionary computation context [KT01].

The SOC PSO introduces for each particle i a critical value C_i (the *criticality* of the particle), which is initialized to $C_i(0) = 0$. The criticality is in some way associated to the diversity of the swarm: in fact its value is increased when two particles become closer than a fixed threshold θ_{SOC} to each other. To balance this mechanism, at each iteration the criticality is reduced by a percentage value ρ_{SOC}:

$$C_i(t+1) = (1 - \rho_{SOC})C_i(t). \tag{2.19}$$

When a particle's criticality becomes larger than a threshold C_{MAX}, the particle is relocated to a different position in the search space, and its criticality is set to:

$$C_i(t+1) = C_i(t) - C_{MAX}. \tag{2.20}$$

Particle relocation has the effect of reducing crowding in over-populated regions, thus promoting diversity in the swarm. The inertia weight can also be modified by introducing a dependence on the criticality of the particle:

$$w_i = 0.2 + 0.1 \cdot C_i. \tag{2.21}$$

In this way, particles in densely populated regions of the search space won't experience the typical cooling off effect associated with low inertia weight.

2.4 Swarm topology

In the previous sections, we have described swarms in which particles had access to the accumulated knowledge of the whole population, as they were attracted by the *global* best solution, *gbest*. However this is not by far the only possible choice. In the following we will discuss about different ways particles can interact with each other. Their interactions, in fact, can be modeled after what we will call different *social structures*.

The study of the social behavior of many species was one of the starting points of the research which led to the development of the particle swarm algorithm. Thus, it is evident that the way particles interact is one of the core aspects of the algorithm. As we will discuss in brief, by modifying the social structure of the swarm, one can deeply influence its dynamics, with important consequences when the swarm simulation is applied as an optimization algorithm. Moreover, we will see how operating on the social structure that underlies particles' interactions provides for a natural approach to niching in the PSO algorithm.

2.4.1 Social networks

A general approach to define alternative social structures consists in introducing the concept of *neighborhood topology*. The *neighborhood* of a particle is defined as the specific subset of the population, whose particles it can communicate with. The mathematical binary relation which defines the neighbors of each particle in a swarm

constitutes the *neighborhood topology* of the swarm. A common way to represent the neighborhood topology of a swarm is by considering its graph, in which each particle is a vertex and is linked by an edge towards each of its neighbors.

Although in theory it is possible to have a directed neighborhood graph, this is very uncommon. In fact two particles usually interact reciprocally, so that the neighborhood relation is symmetrical and the corresponding graph is undirected. Another typical characteristic of the neighborhood graph is that in most cases it is connected, that is there exists a path of adjacent edges linking any two given particles. This means that every particle, although indirectly, can be influenced by any other particle in the swarm. Exceptions to this rule are topologies consisting of multiple isolated sub-swarms, which are used for multimodal function optimization (see Chapter 3).

It is important to note that the neighborhood topology connects particles in a way that is totally unrelated to their position in the search space. The neighbors of a particle can be in regions of the search space which are at arbitrary distances from one another. Moreover, whilst the position of particles changes throughout the simulation, as they fly over the search space, the network topology usually remains fixed, continuing to connect particles even if they happen to be far away in space.

In practice, it is quite straightforward to make the shift from the particle swarm model we discussed in the previous chapter to a more general model with a generic neighborhood topology. In fact, the major change in the algorithm is the replacement of the single *gbest*, the global best solution in the swarm, with a *nbest*, a *neighborhood best*, which is specific to each particle and represent the best solution found among its neighbors. Thus all the equations will remain essentially unchanged, except for the substitution of *gbest* with *nbest*. The only additional burden is that we must maintain a different *nbest* for each particle, instead of a single value for the whole swarm.

2.4.2 Information flow

The role of the neighborhood topology is to regulate the information flow among particles in the swarm. In fact, in the PSO algorithm, particles communicate essentially by sharing with their neighbors the high fitness solutions they previously located.

Thus, the communication takes places along the channels defined by the neighborhood structure. The different topologies regulate in different manners the

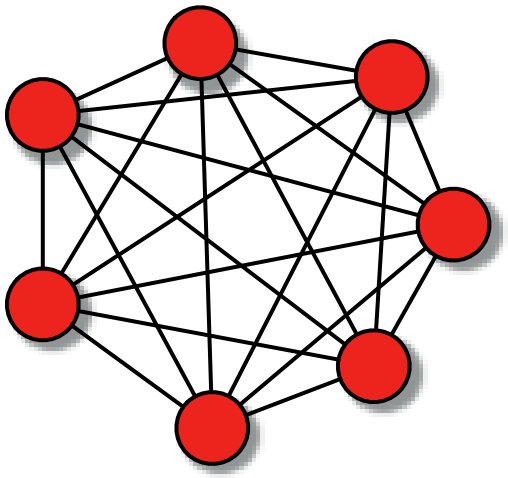


Figure 2.2: Fully connected graph: each particle's neighborhood is the whole swarm (*gbest* Particle Swarm).

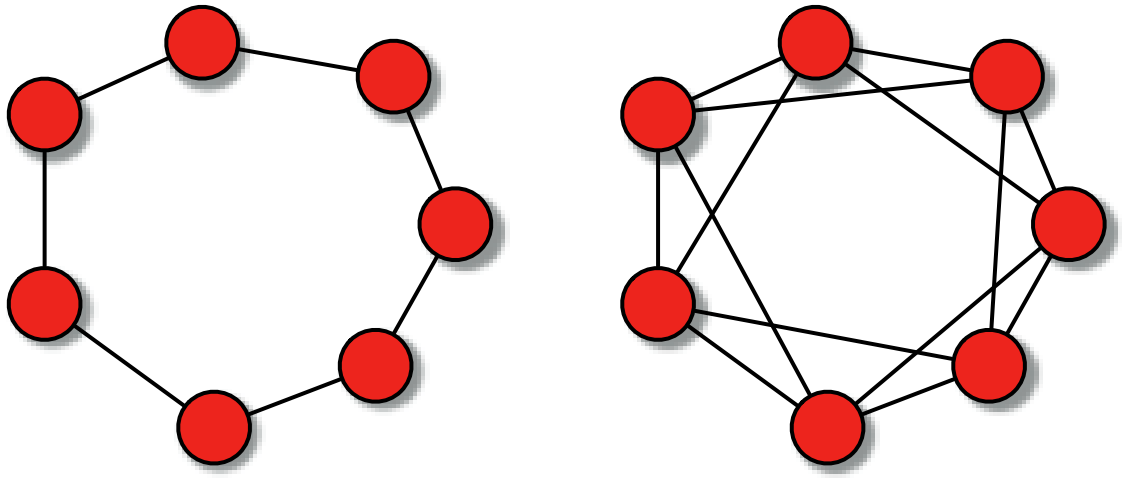
flow of information between particles. On a dense topology particles would have many neighbors, so the knowledge of a good position would be rapidly shared. Conversely, on more sparse structures, the information would spread at a slower rate.

The intensity of the information flow has a noteworthy influence on the dynamics of the particle swarm. In the following, we will present the most used neighborhood topologies, along with some discussion about their effects on the PSO algorithm.

2.4.3 Neighborhood topologies

The swarm considered in the previous chapter, in which a global best position is shared among all particles, also called the *gbest* particle swarm, is the special case whose topology is a fully connected graph, so that each particle's neighborhood is the whole swarm (Figure 2.2). At the other extreme would be the case in which we have no edges in the graph: a particle's neighborhood would thus be empty: this is what has been called the *cognition only* model [Ken97]. However, in this case we would no longer have a *swarm*, but a group of isolated particles with no interactions, which will each perform a local search as a sort of stochastic hill-climbers [RN95].

A more common variation is the so-called *lbest* particle swarm, in which particles are modeled to have only local interactions [EK95]. The topology structure is a regular ring lattice with k edges per vertex, so that each particle is influenced by k neighbors. The best solution in a particle's neighborhood is called *lbest* and its



(a) $k = 2$ (b) $k = 4$

Figure 2.3: Regular ring lattice (*lbest* Particle Swarm).

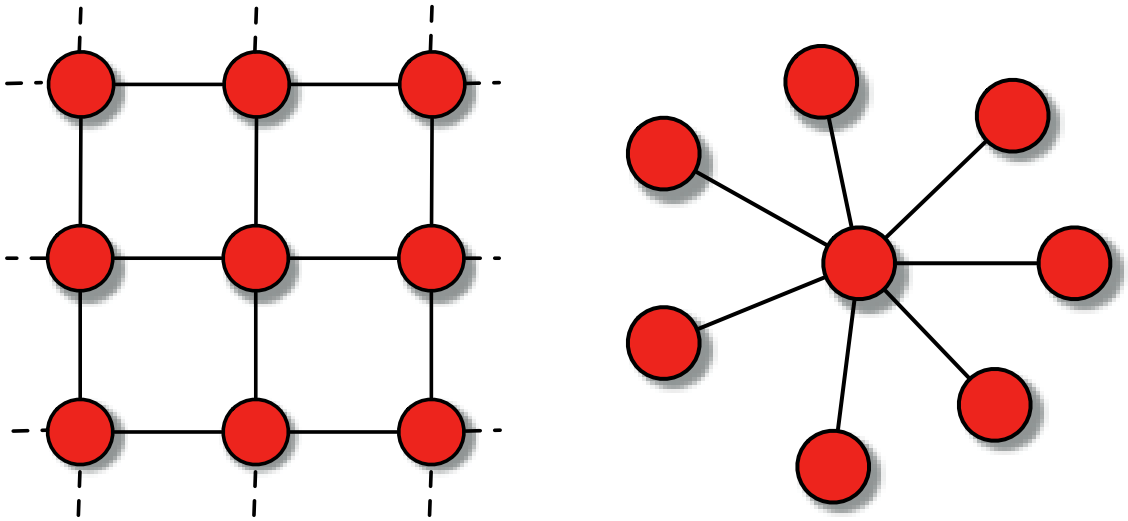
index $l(i)$ is a function of the index i of the particle as in Equation (2.22).

$$l(i) = \arg\min_j f(\mathbf{p}_j(t+1)), \quad j = (i \pm k') \bmod N, \quad k' \leq \frac{k}{2} \tag{2.22}$$

The velocity updating formula is thus only slightly modified, with the substitution of the global index g with the local one $l(i)$. In fact the *gbest* case can be obtained considering $k = N - 1$.

Commonly used *lbest* cases are $k = 2$ (see Figure 2.3a) and $k = 4$ (Figure 2.3b). The former, in particular, results in a swarm in which individuals are affected by only their immediately adjacent neighbors. In such a swarm, it might be the case that a segment of the population converges on a local optimum, while another converges on a different optimum or keeps searching. However, as influence slowly propagates along the ring, and if an optimum really is the best found by any part of the population, eventually all the particles will be pulled towards it.

First experiments with PSO used mainly the *gbest* and *lbest* versions, the latter usually with $k - 2$. The *gbest* topography allows direct communication between all particles, resulting in rapid convergence of the whole population towards the best solution found in the early iterations. However, this can prevent exploration outside of locally optimal regions. The *lbest* topography, instead, limits the flow of information between particles, as it is buffered by the presence of intermediaries. Thus, *lbest* is thought to perform better on multimodal functions, since different



(a) von Neumann (b) Star

Figure 2.4: The von Neumann topology - a two-dimensional lattice - and the star topology - one central node connected with all the others.

group of particles can explore different regions.

Aside from this first two examples, many other topologies have been proposed and studied. Suganthan [Sug99], for example, proposed a swarm in which neighborhood relations extend over time, in order to increase cooperation towards the end of a run. In fact the topology used is the same as *lbest*, but with k increasing from 0 to $N-1$.

Other interesting social structures are the star and the von Neuman topologies (see Figure 2.4). The former represents the common organizational pattern with a single *chief* which communicate with all the other individuals. Thus each particle has only one neighbor, except for the central one whose neighborhood extends over the whole swarm (Figure 2.4b). The latter, named von Neumann's architecture after its use in cellular automata pioneered by John von Neumann, consists of a two-dimensional lattice, in which a particle's neighbors are above, below, and on each side of it (Figure 2.4a).

Comparison

The effects of employing different topologies on the performance of the PSO algorithm have been considered in a number of studies [Ken99, KM03, MKN03]. Here we report briefly the results of a significant study by Kennedy and

Mendes [KM02].

The study considered the most used topologies we presented in the previous section, *gbest*, *lbest*, star, and von Neumann. Moreover, a collection of structures were randomly generated following some criteria derived from the analysis of the information flow in social networks [WS98]. In particular, two factors were considered:

k – the number of neighbors of each particle;

C – the number of neighbors a particle has in common with other neighbors (it accounts for the level of *clustering* in the network).

The random graphs were generated by varying the average values and the standard deviation for k and C. The authors' hypothesis was that populations with rather heterogeneous structures could provide for a good combination of the *gbest* and *lbest* versions, resulting in both quick convergence and good performance on multimodal functions.

The performance analysis was conducted on a set of five commonly used benchmark functions: Sphere, Rastrigin, Griewank, Rosenbrock, and Shaffer's f6. In particular, the following performance measures were considered:

1. standardized performance, the mean standardized best function result after 1000 iterations;

2. proportion, the average proportion meeting success criteria by 10000 iterations;

3. iterations, the median number of iterations required to meet the criteria.

As expected, the conclusions of the study strongly depend on the performance measure considered. By considering standardized performance and proportion, the best topologies were among those randomly generated with $k = 5$, thus confirming that a quite low number of connections helps to avoid premature convergence. By considering the number of iterations to reach the success criteria, the best ones were those with $k = 10$ and the *gbest* topology: highly connected topologies are the quickest to converge – but they are also prone to get stuck in local optima.

The results obtained with the special topologies were also quite interesting. The star topology showed among the worst performance considering all three measures, thus censoring the use of a centralized architecture. The *gbest* topology confirmed

to be characterized by a rapid convergence, thus showing good results as number of iterations, but was often unable to find the global optimum, so its performance with the other measures were quite low. The *lbest* topology showed to be better than *gbest* in avoiding local optima, but had very poor performance compared with other structures. Finally, the von Neumann topology proved to have the best overall performance, with high success rate and quite good convergence speed, so it is the one the authors recommend to particle swarm researchers.

2.4.4 Spatial interactions

Another area of research on the particle swarm, which is rather orthogonal to the social structures, consists in introducing interactions between particles which are close in the solution space, rather than connected in the topological graph.

Work in this direction has been carried out by Peram, Veeramachaneni, and Mohan in [PVM03, VPMO03]. They propose a new algorithm in which particles use the ratio between the fitness and the distance of other particles to determine the direction in which their position needs to be changed. The resulting algorithm, known as Fitness-Distance-Ratio-PSO, tends to avoid convergence at early stages of particles evolution, thus could be well-suited to search for global optima in difficult multimodal optimization problems.

In the same direction is the approach proposed by Krink, Vesterstrom, and Riget in [KVR02], which provides particles of a *spatial extension*. Thus the algorithm is enhanced by *collision detection* and *bouncing* mechanisms, aimed at avoiding particles clustering near local optima, and favoring diversity.

Chapter 3

Niching

In discussing the introduction of the Particle Swarm Optimization algorithm in the previous chapter, we focused on the obvious goal in the design of an optimization algorithm: that it should be able to find a single optimal solution to a problem. In fact this is common among most techniques employed in optimization problems, from simple hill-climbing to evolutionary algorithms.

However, there are many interesting scenarios in which the problem one wishes to optimize is multimodal, that is it allows for multiple optimal or semi-optimal solutions. In these cases finding only one solution may not be the best strategy. Ideally, an optimization algorithm should find *all* the optimal solutions, then these solutions can be used in different ways, which are dependent on the nature of the problem and on the specific application field. In some cases a single solution can be chosen from the set, according to some additional criteria which were not included in the optimization. In other situations, as for example in the field of pattern classification, it may be possible to combine different solutions to build more complex – and effective – ones [Kun04].

In the following we will focus our attention on the optimization of multimodal functions, since it is the more formal and general way of dealing with optimization problems with multimodal domains. We will discuss the general techniques employed for the optimization of multimodal functions. Then we will focus on the field of evolutionary algorithms, where multimodal functions are dealt with by introducing the concept of *niching*. Finally, we will present some approaches to niching in the field of Particle Swarm Optimization.

Table 3.1: A classification of functions based on multimodality.

	Without local optima	With local optima
Single global optimum	Unimodal functions	Multimodal type 2
Multiple global optima	Multimodal type 1	Multimodal type 3

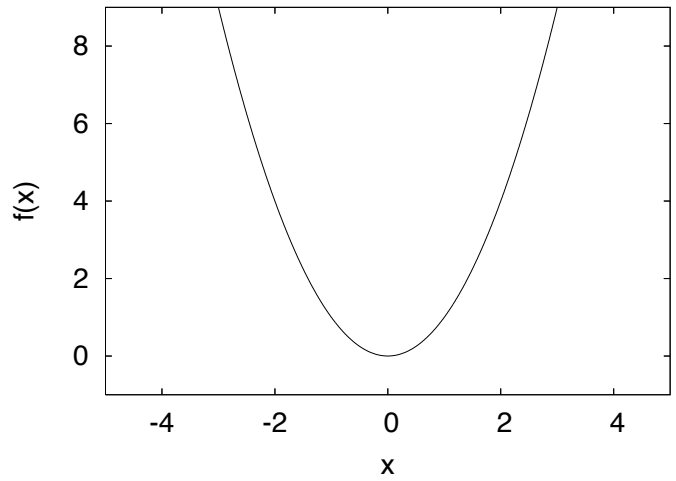


Figure 3.1: Function $f(x) = x^2$, an example of a simple unimodal function, with a unique global minimizer at $x = 0$.

3.1 Multimodal function optimization

Optimization problems can be divided in different classes depending on whether they allow for a single or for multiple solutions. The first class is that of *unimodal* problems. An optimization problem is considered to be unimodal when its objective function $f(\mathbf{x})$ has a single global optimum point $\mathbf{x}^*$ and no local optima (see Section 2.1.2). Figure 3.1 shows a really simple example of a unimodal function.

Conversely, optimization problems which allows for multiple solutions are considered *multimodal*. Thus, the objective function has multiple optimizers, although a further differentiation can be made depending on whether they are local or global optimizers:

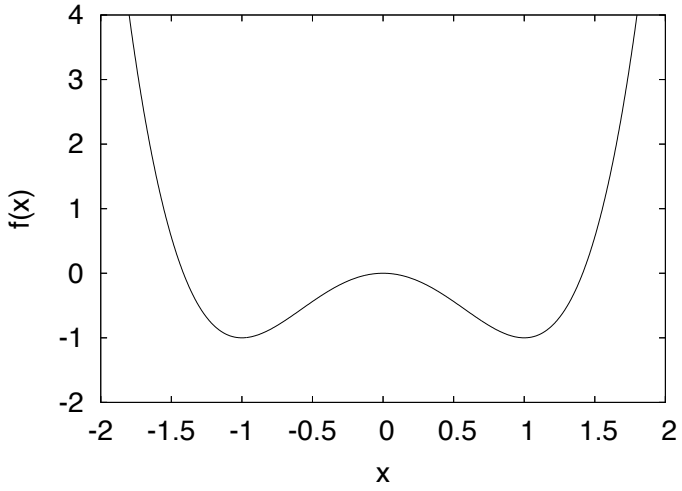


Figure 3.2: Function $f(x) = x^4 - 2x^2$, an example of a multimodal function with two global minimizers at $x = -1$ and $x = 1$.

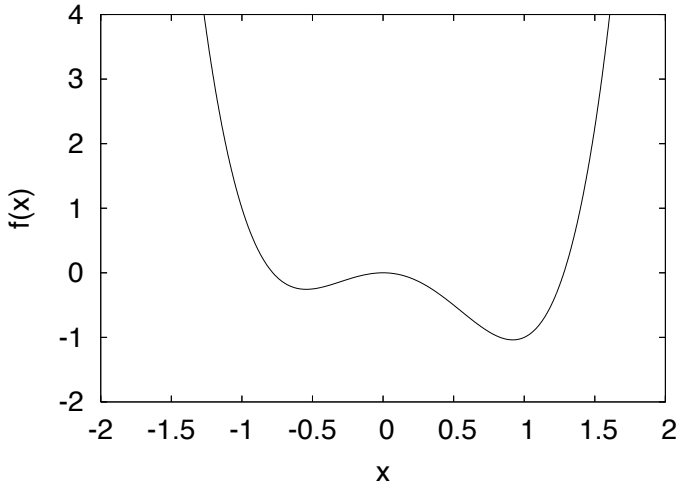


Figure 3.3: Function $f(x) = 2x^4 - x^3 - 2x^2$, an example of a multimodal function with a local minimizer at $x \simeq -0.544$ and a global one at $x \simeq 0.919$.

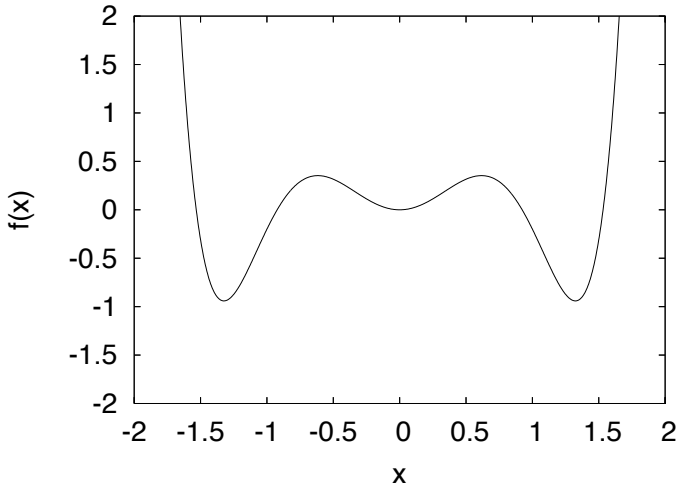


Figure 3.4: Function $f(x) = x^6 - 3.2x^4 + 2x^2$, an example of a multimodal function with two global minimizers at $x \simeq \pm 1.324$ and a local one at $x = 0$.

1. A particular kind of multimodal functions is characterized by the presence of multiple global optimizers and no local ones. An example function is plotted in Figure 3.2: its two optimizers have the same function value, so they are both global. Optimizing a function of this kind usually means searching for *all* the global optimizers.

2. Another kind is constituted by functions with a single global optimizers, but additional local ones. In particular, the one plotted in Figure 3.3 has a local minimizer besides the global one. The goal in the optimization of this kind of function can be twofold. One can either look for all the optimizers, regardless of them being local or global ones, or focus just on the single global optimizer.

3. Finally, the most complex kind of multimodal functions includes those with multiple global optimizers *and* also local ones. In particular the example plotted in Figure 3.4 shows a function with a local minimizer situated just between two global minimizers. This is a typical case of a *deceptive* local optimum, as many optimization algorithms can be fooled to converge on it, missing one or both of the global optimizers.

As already stated, when trying to optimize a multimodal function, very different

problems arise in each of these cases. Optimization algorithms applied to functions with a global optimizer and many local optimizers have usually the goal to locate the single global optimizer. Thus their main problem is avoid deception by the other optima of the multimodal function, which would otherwise result in premature convergence to a suboptimal solution. Most optimization techniques are designed to deal with multimodal functions of this kind. In fact, they usually assume that there exists only a single best solution in the search space and put effort into isolating it from other spurious solutions.

In the other cases, when the function has many optimizers corresponding to the same value, the typical goal of an optimization algorithm is to locate *all* of them, eventually considering also the local solutions. However, standard techniques are generally not designed to fulfill such a goal, and will usually either favor a single solution, or get confused by the multiple possible solutions and fail to converge to a single one. Thus new specific algorithms need to be employed or the standard ones need to be modified to be more effective on multimodal functions.

In this thesis we will focus on optimization algorithms which have the goal to find multiple solutions, as we are interested in dealing with general multimodal functions. However, it is interesting to note that even when facing a multimodal function of the first kind and looking for the single optimal solution, an optimization algorithm which looks for multiple solutions can still be useful. In fact, the methods used to diversify the search and to locate different optimizers can also help in avoid premature convergence on local solutions. Thus, although the algorithms we will discuss are all designed to perform a search for multiple solutions, they can also be applied effectively when the problem is to discover the global optimum.

3.2 Niching techniques in genetic algorithms

Niching techniques are modeled after a phenomenon in nature where animal species specialize in exploiting different kinds of resources, resulting in several species coexisting in the same environment [Mah95a, Hor97]. The introduction of this specialization, or niching, in a search algorithm allows it to divide the space in different areas and search them in parallel. The technique has proven useful when the problem domain includes multiple global and local deceptive optimal solutions.

Niching was introduced in the field of Genetic Algorithms (GA), although it can be quite easily generalized to the whole context of evolutionary algorithms. Niching

methods can reduce the effect of the genetic drift operated by the selection operator in the standard GA. In fact they maintain diversity in the population and allow the algorithm to investigate multiple peaks in parallel. Moreover they help in preventing the premature convergence which can trap the algorithm in a local optimum of the search space.

Genetic algorithms are based on the principles of genetics and basically try to imitate natural evolution. However standard GAs tend to rapidly converge to a population of very similar individuals, whilst natural evolutionary processes maintain a variety of species which occupy different ecological niches. In a natural environment, in fact, the competition among individuals of the population can develop upon the exploitation of different kind of resources, leading to the evolution of different species. In a biological context, a niche is a subspace of the environment supporting different types of life. A species is a group of individuals sharing most biological features and capable of interbreeding among themselves, while generally unable to breed with individuals belonging to other groups. Thus competition, which is at the core of the evolutionary process, develops along two levels. The first level of competition is among individuals in the same group, that is inside a single species. The second one is among different species contending the resources of the same environmental niche.

The concepts of natural niching are implemented in genetic algorithms by implicitly or explicitly dividing the population in different species which tend to occupy different niches, that is different regions of the search space, possibly leading to different optima. Therefore, in the context of the implementation of niching methods we will indifferently talk about species and niches.

An important variety of niching methods have been reported in the literature, including sequential niching [BBM93], immune systems [FJSP93], speciation with implicit fitness sharing and co-evolution [DY97], ecological GAs [Mah95a, Dav91], and crowding schemes. Our first analysis regards the special class of sequential niching methods.

3.2.1 Sequential vs parallel niching

A clear distinction can be made amongst algorithms for multimodal function optimization, which depends on whether they search for different solutions sequentially or in parallel. The first choice logically allows for the slightest modifications to the original optimization algorithm. In fact, an algorithm which

looks for a single solution can be applied repeatedly to discover different optima, although usually some method is introduced to avoid or at least limit rediscovering already known solutions. The second choice typically involves a deeper re-thinking of the optimization algorithm, as it has to search in parallel for different solutions. However, this can also result in an advantage, since this methodology allows to exploit the information relative to the multiple optima being discovered.

In the niching framework, sequential methods, known as *sequential niching*, probably occupy a special position. In fact, it can be argued that a niching approach inspired by the natural differentiation of species in the environment must operate in parallel, rather than sequentially. However, in literature both the approaches are considered kinds of niching.

The most successful sequential niching method is probably the one proposed by Beasley, Bull, and Martin [BBM93]. The method works by running a simple GA multiple times and maintaining the best solution of each run off-line. To avoid converging to the same optimum multiple times, the algorithm depresses the fitness landscape at all the points within a fixed radius σ of previously discovered solutions. In this way, at each run the GA is applied to a modified function, where the location of an already known solution of the original function is no more an optimum point, that is it has been *derated*.

The supposed advantages of this technique lie in its simplicity, as no major modification is required to the standard GA, and in the smaller population size required with respect to a parallel approach, since only one solution is searched for during each run. Actually, sequential niching proved to be an effective niching approach only on quite simple functions. In more complex cases, in fact, the deration process can cause many problematic issues [Mah95b]:

- Derating the fitness landscape near previously located peaks can create new false optima;
- or it can hide other actual optima which were too close to already discovered ones.
- Moreover, especially in advanced runs, when many optima have already been found, the fitness landscape will contain many derated regions, with plateaus and small ridges, which can raise the difficulty to locate new solutions by a high degree.

In the following discussion, we will focus our attention on parallel niching methods or classes of methods which have been most commonly used or have proved to be most successful: Fitness Sharing, Crowding, and Clearing.

3.2.2 Fitness sharing

The most known and used method to add niching capabilities to a genetic algorithm is probably the sharing method. It was originally introduced by Holland [Hol75] and improved by Goldberg and Richardson [GR87].

The core concept behind the fitness sharing method is to model environmental niches with limited resources by making individuals in the same niche *share* their fitness. From a different perspective, it can be seen that in fact the introduction of fitness sharing modifies the search landscape by reducing the payoff in densely populated regions. In practice, the sharing effect is obtained by reducing the fitness of each individual in function of the number of similar individuals in the population. Thus, in this context, we will define the shared fitness f_i' of an individual i with original fitness f_i as

$$f_i' = \frac{f_i}{m_i} \tag{3.1}$$

where m_i is the niche count which roughly measures the number of individuals with whom the fitness is shared. Although the final goal of the niching algorithm is to dynamically form species which adapt to environmental niches, we have no knowledge of the exact number of individuals in a niche at every given time. Thus the niche count must be estimated using a measure of similarity between individuals: m_i is calculated by summing a sharing function over all members of the population

$$m_i = \sum_{j=i}^{N} sh(d_{ij}) \tag{3.2}$$

where N denotes the population size and d_{ij} represents the distance between the individual i and the individual j. The sharing function sh measures the similarity level between two population elements. It is equal to 1 if the elements are identical, to 0 if their distance is higher than a threshold of dissimilarity, and it is in the interval $]0, 1[$ for intermediate levels of dissimilarity. The most widely used sharing function is given as follows:

$$sh(x) = \begin{cases} 1 - \left(\frac{x}{\sigma}\right)^{\alpha}, & \text{if } x < \sigma \\ 0 & \text{otherwise} \end{cases} \tag{3.3}$$

where σ denotes the threshold of dissimilarity and α is a constant parameter which regulates the shape of the sharing function. α is commonly set to one with the resulting sharing function referred to as the triangular sharing function [Gol89]. The threshold of dissimilarity σ is also known as the distance cutoff or the niche radius and plays a very important role in the niche formation process. In fact, it sets the maximum range for individuals to influence their reciprocal niche counts, thus establishing the radius of the resulting niches. As we will discuss in the next section, the need to set *a priori* the niche radius is probably one of the most notable limitations of fitness sharing methods.

The distance d_{ij} between two individuals i and j, and its related similarity metric can be based on either genotypical or phenotypic similarity. Genotypical similarity is related to the specific encoding used by the genetic algorithm, typically a bit-string representation, and is generally the Hamming distance. Instead, phenotypic similarity is directly linked to real parameters of the search space. It can be the Euclidian distance for instance. There is some evidence [DG89] that sharing based on phenotypic similarity gives slightly better results than sharing with genotypical similarity, due to decreased noise in the decoded parameter space.

When implementing fitness sharing, particular attention must be paid also to other elements of the evolutionary process, such as the selection mechanism and the genetic operators. Any selection method can be used in conjunction with sharing, however the choice can either increase or decrease the stability of the algorithm, as it influences the maintenance of multiple niches in the population. Two widely used methods are Stochastic Remainder Selection (SRS) and Stochastic Universal Selection (SUS) which help to reduce bias in the selection algorithm [Bak87]. Tournament Selection (TS) can also be used, although with some modifications to help promoting stability. In particular, Oei, Goldberg, and Chang [OGC91] propose a tournament selection with continuously updated sharing, which calculates the shared fitness with respect to the new population being filled.

Similarly to the choice of the selection method, also the design of the recombination operators must take into account the need to stably maintain multiple subpopulations. The main problem is that recombination operators such as crossovers are subject to the formation of *lethals*, a phenomenon also known as crossover disruption. Lethals are individuals with very low fitness originated

by the recombination of parents belonging to different niches. The most used techniques to reduce the formation of lethals are based on mating restriction schemes [DG89, YG93, MS96].

Limitations

Fitness sharing is one of the most popular niching techniques for GAs, as it both favors the formation of stable niches and tends to perform search in unexplored regions of the space. However, it has some limitations which can hinder its effectiveness. In particular we can point out as a serious limitation the need to set in advance the niche radius, which can actually prevent the application of fitness sharing to a whole kind of multimodal problems. In fact, setting the dissimilarity threshold σ requires a priori knowledge of how far apart the optima are in the search landscape. When dealing with real optimization problems, however, in general no or very few information about the search space and the distance between the optima is available. Moreover, fitness sharing imposes that σ is the same for all individuals, thus assuming that all peaks are nearly equidistant in the domain. Again, in real world problems this could not at all be the case.

When trying to apply the fitness sharing technique to a multimodal function for which we cannot correctly estimate the distance between peaks, or whose peaks are not even roughly equidistant, the algorithm will often fail to maintain all desired peaks. Various empirical formulas have been proposed to set the dissimilarity threshold but this problem remains probably the major flaw of the method [DG89, Mah95a].

Another limitation of the sharing scheme is its computational cost. In fact, it is very expensive because it requires the computation of niche counts of complexity $O(N^2)$ per generation. Among the methods developed to reduce computational complexity and also increase sharing effectiveness we can cite clustering analysis [YG93] and dynamic niching [MS96]. However, in many domains the computational time to obtain the fitness of individuals dominates the computational cost of comparisons. In that case, standard sharing can be implemented with only a small increase in the computational requirements.

Fitness scaling

A particular variation of fitness sharing which tries to improve efficiency is fitness scaling [Gol89]. A scaled shared function increases differentiation between optima

and reduces deception, as it lowers the attraction towards local optima [GDH92, DY95], whilst making the global optima more attractive than the surrounding regions of the space. A common technique to scale the fitness function is to use a power scaling; Equation (3.1) can thus be modified as in the following:

$$f_i' = \frac{f_i^\beta}{m_i}, \tag{3.4}$$

where m_i is the niche count as defined in Equation (3.1). The remaining problem is the choice of an appropriate β for a given objective function. If the power of the scaling function is too high, the predominance of fitness scaling can prevent the reduction of genetic drift by the sharing method. The domination of "super-individuals" in the population can cause the niching GA to converge prematurely. On the other hand, if the power of the scaling function is too low, differentiation between optima can be insufficient. This can hinder a perfect detection of the optima by the sharing method. The compromise in the choice of the scaling power is directly related to the accurate balancing between exploration and exploitation necessary to all global stochastic optimization methods. To prevent premature convergence and increase the efficiency of the sharing method, the scaling power β can be adapted during the search, as recommended in [DY95].

3.2.3 Crowding

A different approach to promote the formation of niches in the population is based on crowding methods. As we have seen in the previous section, fitness sharing acts indirectly, by modifying the landscape of the search space penalizing too crowded regions. Instead, crowding methods directly operate on the population, inserting new individuals by replacing similar ones. Various flavors of crowding techniques have been developed. Here we examine the most notable ones.

Standard crowding

In DeJong's crowding [DJ75], only a fraction of the global population specified by a percentage G (generation gap) reproduces and dies at each generation. In this crowding scheme, an offspring replaces the most similar individual (in terms of genotypical comparison) taken from a randomly drawn subpopulation of size CF (crowding factor) from the global population.

Other methods very similar to De Jong's standard crowding are *preselection* schemes, previously introduced by Cavicchio [Cav70]. Preselection schemes work by letting children directly replace the parent individuals which produced them. These methods can be classified as a kind of crowding, since the parents which are replaced are in fact usually similar to their children.

Although crowding was introduced to limit the genetic drift towards a uniform population, thus favoring diversity, these characteristics seemed likely to provide also for a good niching strategy. In fact, the way crowding works is somewhat similar to how natural populations effectively maintain separate species.

However, when applied to multimodal function optimization, standard crowding never obtained good results [DG89, Mah95a], as it failed to maintain more than two niches even on quite simple functions. The reasons for this behavior can be found in the high stochastic error in the replacement of population members. In fact, since the new individuals pending insertion are compared to those in a set which is chosen randomly, it is quite likely that a member of a niche is replaced by an individual belonging to a different niche. In the long run, replacement errors bring in again genetic drift, which do not allow the algorithm to maintain multiple niches.

Deterministic Crowding

Mahfoud's Deterministic Crowding (DC) improved standard crowding by introducing competition between children and their parents [Mah95a]. This behavior is inspired by that of the preselection schemes cited above. However, other aspects of the algorithm make it stand apart from previous crowding methods and effectively perform as a niching algorithm.

As described in Algorithm 3.1, at each generation the DC algorithm randomly pairs all population elements. Each pair undergoes crossover, and possibly mutation or other genetic operators, yielding two children. Then the children individually compete with one of the two parents, according to a similarity rule, to be included in the new population. This behavior differs from that of preselection schemes for two main reasons:

1. The selection mechanism. In preselection schemes, selection is usually fitness–proportionate, and is performed before mating parents (*up-front* selection). It is in fact the classical way of applying a selection pressure in a standard GA, which is completely unaware of niches. In DC, instead, there is no up-front selection: *all* individuals in the population are used in mating. Thus they

Algorithm 3.1 Mahfoud's Deterministic Crowding pseudocode.

```
1: procedure DETERMINISTIC CROWDING
2:     for T generations do
3:         for n/2 times do
4:             Select two parents, p_1 and p_2, randomly without replacement
5:             Apply crossover, yielding c_1 and c_2
6:             Apply mutation (and possibly other operators), yielding c'_1 and c'_2
7:             if d(p_1, c'_1) + d(p_2, c'_2) ≤ d(p_1, c'_2) + d(p_2, c'_1) then
8:                 if f(c'_1) > f(p_1) then
9:                     Replace p_1 with c'_1
10:                end if
11:                if f(c'_2) > f(p_2) then
12:                    Replace p_2 with c'_2
13:                end if
14:            else
15:                if f(c'_2) > f(p_1) then
16:                    Replace p_1 with c'_2
17:                end if
18:                if f(c'_1) > f(p_2) then
19:                    Replace p_1 with c'_2
20:                end if
21:            end if
22:        end for
23:    end for
24: end procedure
```

always contribute to generate new offsprings and only then they compete for a place in the new population (*replacement* selection). The resulting selection pressure does not inhibit the formation of niches.

2. The tournament. In a preselection scheme, only one of the children is chosen to compete its parents, while the other is discarded. If the new offspring has a superior fitness than that of at least one parent, it replaces the worst parent. However, this behavior was one of the causes for replacement errors. In DC this scheme was modified: each of the two children compete with one of the parents, and the winners of the two tournaments gain a place in the new population. The choice of which child competes with which parent is made according to a similarity measure, in order to favor competition of individuals which are close in the search space. In this regard, DC can be applied both using a genotypical and a phenotypic similarity measure, although the latter usually provides better results.

Differently to standard crowding, the DC algorithm proved to be an effective niching technique which can be applied to multimodal function optimization problems. Thanks to the implicit elitism of the tournament mechanism it adopts, DC can avoid the problem of crossover disruption, which is indeed very common in GA on multimodal domains. In fact, when an offspring results from the crossover of parents in different niches, and its fitness is lower than its parents, it loses the competition against them and is simply discarded. Another advantage of DC over standard crowding, and also fitness sharing, is its low computational complexity. In fact, with two distance comparisons per set of tournaments and $N/2$ sets of tournaments per generation, the resulting order of complexity of DC is $O(N)$.

Restricted Tournament Selection

The Restricted Tournament Selection (RTS) resulted from an adaptation of standard tournament selection to deal with multimodal function optimization [Har95]. RTS works by initially selecting two elements from the population and applying to them crossover and mutation. After this recombination process, a random sample of CF individuals is taken from the population as in standard crowding. Each offspring competes only with the closest element from this sample and the winners are inserted in the new population. The order of complexity of RTS is $O(CF \cdot N)$. It can vary from $O(N)$ to $O(N^2)$ according to the crowding factor value CF. Experiments show

that RTS is indeed an effective niching method, and can outperform DC on many test functions [SK98]. However, its main limitation is probably the requirement of the crowding factor parameter, which is very problem-specific, and of difficult estimation.

3.2.4 Clearing

The clearing method introduced by Petrowski [Pet96], is based on the same concept of limited resources in the environment which inspires fitness sharing methods. The fundamental difference between the two methods is that instead of sharing the resources between all individuals in a niche as in fitness sharing, clearing assigns them only to the best members of the niche.

The clearing procedure is based on the assumption that the subpopulation in each niche contains a dominant individual: the one that has the best fitness. If an individual belongs to a given subpopulation, then its distance from the dominant is less than a given threshold σ: the clearing radius. The basic clearing algorithm preserves the fitness of the dominant individual while it resets the fitness of all the other individuals of the same subpopulation to zero. Thus, the clearing procedure fully attributes the whole resource of a niche to a single individual: it is a winner-takes-all strategy rather than a sharing strategy as in sharing methods.

In practice, the capacity q of a niche specifies the maximum number of elements that each niche can accept. Thus, clearing preserves the fitness of the q best individuals (dominant individuals) of the niche and resets the fitness of the others that belong to the same subpopulation (dominated individuals). The pseudocode of the clearing procedure is given in details in Algorithm 3.2.

Clearing is often coupled with elitist strategies to preserve the best elements of each niche along generations. The order of complexity of the clearing procedure is $O(k \cdot N)$ where k is the number of niches maintained during the search. Although clearing proved to be a quite effective niching technique [SK98], its major drawback is the same as that of fitness sharing methods: the difficulty of estimating a fixed niche radius.

Species Conserving Genetic Algorithm

A different approach was proposed by Li *et al.* which focuses on determining and conserving species: the Species Conserving Genetic Algorithm (SCGA) [LBPC02]. A species in SCGA is the subpopulation occupying a niche in the environment, and

Algorithm 3.2 Petrowski's Clearing pseudocode.

```
1: procedure CLEARING
2:     Sort individuals based on decreasing fitness values
3:     for i = 1 → n do
4:         if f(p_i) > 0 then
5:             dominants := 1
6:             for j := i + 1 → n do
7:                 if f(p_i) > 0 and d(p_i, p_j) < σ then
8:                     if dominants < q then
9:                         dominants := dominants + 1
10:                    else
11:                        f(p_j) := 0.0
12:                    end if
13:                end if
14:            end for
15:        end if
16:    end for
17: end procedure
```

it is identified by a procedure very similar to clearing. But instead of operating on the fitness of the dominant individuals, called the species seeds, it applies a form of elitism on them. At each generation, the species seeds are saved in a separate location, while the population undergoes the typical genetic operators. Then the seeds are reintroduced in the new population, except if it already contains other individuals of the same species.

The peculiarity of this approach is that it does not modify either the genetic operators, or the selection mechanism. Rather, it performs niching by introducing a species conserving elitist strategy in an otherwise standard GA. Our interest in the species conserving approach, however, is mostly due to the fact that it was later applied also to the PSO, as we will discuss in Section 3.3.4.

3.3 Niching techniques in PSO

Like most optimization algorithms, the Particle Swarm algorithm was designed with the goal of finding a single global optimal solution. In fact most of the research on PSO focused on its ability to avoid local optima in multimodal functions of the first kind we described, i.e. with one global optimum among many local ones. How it can deal with the goal of finding multiple solutions is quite a different issue.

In the previous section, we have seen how the standard genetic algorithm, which *per se* could not deal with multimodal functions, could be improved by using niching techniques. Given the similarities between the evolutionary and the particle swarm approach, it is natural to consider a parallel development of niching for PSO. There are however some peculiarities of the particle swarm approach which need to be considered.

Niching methods for genetic algorithms modify the way individuals interact with each other, in order to let them specialize on different areas of the search space, instead of converging on a single one. In the evolutionary approach individuals interact rather indirectly, as a combined effect of the selection and recombination operators. In fact, the various niching methods we discussed use different approaches, depending on the specific interaction factor they choose to modify.

The situation with the PSO algorithm is rather different. The particles in the swarm interact in a much more straightforward way than evolving individuals. In fact, the interaction is implemented by sharing knowledge of the search space with

neighbor particles. Thus the neighborhood structure of the swarm is probably the single most relevant aspect to consider in our analysis.

Let's consider for example the *gbest* swarm, that is a PSO in its simplest form. The *gbest* topology let particles interact with the whole swarm. As we have seen in the previous chapter, this provides for very quick convergence towards a *good* region of the space, which often is identified early in the search process. On a function with multiple global optima, this approach most often will simply converge on one of the optima and ignore the others.

Conversely, neighborhood topologies more sparsely connected lead to very different outcomes. A good example is the *lbest* topology, where particles have a very small neighborhood. In this case, the population tends to split in small groups which explore independently the search space. Thus they can actually locate different optima at the same time, resulting in a sort of implicit niching capability. However, since the neighborhoods in the *lbest* swarm overlap, in many cases particles end up to converge on the same optimum nonetheless. In the end, the implicit niching performed by the swarm relies too much upon random events during the running of the algorithm to be an efficient niching approach.

Thus, as pointed out by Engelbrecht *et al.* in [EMP05], the standard PSO must be modified to allow the efficient location of multiple solutions. Notwithstanding the differences between the approaches we will discuss for the particle swarm, with respect to the evolutionary ones, we will still refer to them as niching strategies.

3.3.1 Objective function stretching

Objective Function Stretching, introduced by Parsopoulos *et al.* in [PPMV01b, PPMV01a], was among the first strategies to modify the particle swarm algorithm to deal with functions with multiple optima, in particular multimodal functions of type 2, following our classification in Table 3.1. Specifically, the goal of the authors was to overcome the limitations of PSO caused by premature convergence to a local solution. The stretching approach operates on the fitness landscape, adapting it to remove local optima. Considering a minimization problem, when the swarm converges on a – possibly local – minimum, the fitness of that position is *stretched* so that it becomes a local maximum. In this way, successive iterations of the PSO algorithm will focus on other areas of the search space, leading to the identification of other solutions.

In [PV01], Parsopoulos and Vrahatis further developed the same technique to

use it as a sequential niching approach. For this purpose, the algorithm proceeds by identifying candidate solutions when their fitness is below a fixed threshold value ϵ. Thus when the swarm finds a solution $\mathbf{x}^*$ such that $f(\mathbf{x}^*) < \epsilon$, the fitness is redefined as:

$$H(\mathbf{x}) = G(\mathbf{x}) + \gamma_1 \frac{\text{sign}(f(\mathbf{x}) - f(\mathbf{x}^*)) + 1}{2\tanh(\mu(G(\mathbf{x}) - G(\mathbf{x}^*)))} \tag{3.5}$$

and

$$G(\mathbf{x}) = f(\mathbf{x}) + \gamma_2 \frac{\|\mathbf{x} - \mathbf{x}^*\|(\text{sign}(f(\mathbf{x}) - f(\mathbf{x}^*)) + 1)}{2} \tag{3.6}$$

where suggested values for the parameters are $\mu = 10^{-10}$, $\gamma_1 = 10^4$, and $\gamma_2 = 1$ and the sign function is defined as usual:

$$\text{sign}(x) = \begin{cases} +1 & x > 0 \\ 0 & x = 0 \\ -1 & x < 0 \end{cases} \tag{3.7}$$

The transformation G in Equation (3.6) removes all the local minima located above the detected solution $\mathbf{x}^*$, while H in Equation (3.5) raise the fitness value of the positions close to $\mathbf{x}^*$. The new fitness function H is then used in the successive iterations of the particle swarm algorithm to locate other minima. An additional local search can be performed, using the original fitness function, in the neighborhood of the position $\mathbf{x}^*$ where the candidate solution was found, in order to further refine it and detect whether it was a local or global minimum.

In the same way as other sequential niching approaches, the stretching technique has some advantages, but critical issues. It requires no modifications of the underlying PSO algorithm and actually shows good performance on many multimodal functions. However, the effectiveness of the stretching transformation is not uniform on every function. In fact, in some cases it can introduce false minima, which render this method unreliable [vdB02].

Recently, this approach has been improved by the introduction of other two techniques: Deflection and Repulsion [PV04]. The former is another form of modification of the objective function which removes optima which have been already located. The general definition for the *deflection* technique is:

$$F(\mathbf{x}) = T_1(\mathbf{x}; \mathbf{x}_1^*, \lambda_1)^{-1} \dots T_m(\mathbf{x}; \mathbf{x}_m^*, \lambda_m)^{-1} f(\mathbf{x}), \tag{3.8}$$

where $\mathbf{x}_1^* \dots \mathbf{x}_m^*$ are the optima found in $f(\mathbf{x})$, $\lambda_1 \dots \lambda_m$ are relaxation parameters and $T_1 \dots T_m$ are suitable functions, like the following:

$$T_i(\mathbf{x}; \mathbf{x}_i^*, \lambda_i) = \tanh(\lambda_i \|\mathbf{x} - \mathbf{x}_i^*\|). \tag{3.9}$$

The repulsion technique, instead, modifies the PSO algorithm by introducing a repulsion force in an area surrounding the optimizers which have already been detected. The combination of these techniques seems to alleviate the usual problems of sequential niching approaches.

3.3.2 Niche PSO

In 2002 Brits *et al.* introduced the *nbest* PSO, reportedly the first technique to achieve a *parallel* niching effect in a particle swarm [BEvdB02b]. The *nbest* PSO was in particular aimed at locating multiple solutions to a system of equations, and used local neighborhoods determined by spatial proximity.

The same authors subsequently proposed a new approach which used *sub-swarms* to locate multiple solutions to multimodal function optimization problems: NichePSO [BEvdB02a].

NichePSO, whose pseudocode is given in Algorithm 3.3, maintains a main swarm which can generate a sub-swarm each time a possible niche is identified. The main swarm is trained using the *cognition only* model [Ken97] (see also Section 2.4.3), which updates the velocities of particles considering only their personal best position. Since no *social* component is involved, each particle will perform a local search. Conversely, the sub-swarms are trained using the GCPSO algorithm, described in Section 2.3.3, which ensures convergence to a local optimum.

Niches in the fitness landscape are identified by monitoring changes in the fitness of particles. Specifically, a new sub-swarm is created when the fitness of a particle shows very little change over a fixed number of iterations. Therefore the variance σ_p of the fitness of particle p is tracked over e_σ iterations (usually $e_\sigma = 3$). When $\sigma_p < \delta$, a new sub-swarm is generated containing the particle p and its closest neighbor q such that:

$$q = \arg\min_{i \neq p} \|\mathbf{x}_p - \mathbf{x}_i\|. \tag{3.10}$$

NichePSO has some rules to decide the absorption of particles into a sub-swarm and the merging of two sub-swarms, which basically depends on the measure of the

Algorithm 3.3 NichePSO pseudocode.

```
1: procedure NichePSO
2:     Initialize main particle swarm.
3:     repeat
4:         Perform one step of the cognition only model on the main swarm.
5:         for each sub-swarm S do
6:             Perform one step of the GCPSO algorithm on S.
7:             Update swarm radius.
8:         end for
9:         Possibly merge sub-swarms.
10:        Check for particles to be absorbed.
11:        for each particle p in the main swarm do
12:            if p meets the partitioning criteria then
13:                Create a new sub-swarm with p and its closest neighbor.
14:            end if
15:        end for
16:    until stopping criteria are met.
17: end procedure
```

sub-swarm radius. The radius R_j of a sub-swarm S_j is defined as:

$$R_j = \max_{\mathbf{x} \in S_j} \|\mathbf{x}_{g(j)} - \mathbf{x}\|, \tag{3.11}$$

where $g(j)$ denotes the index of the best particle in the sub-swarm S_j. A particle p is absorbed into a sub-swarm S_j when it *moves* into it, that is:

$$\|\mathbf{x}_p - \mathbf{x}_{g(j)}\| \leq R_j. \tag{3.12}$$

Two sub-swarms S_j and S_i are merged when they intersect, that is when:

$$\|\mathbf{x}_{g(j)} - \mathbf{x}_{g(i)}\| < R_j + R_i. \tag{3.13}$$

This approach was compared empirically and shown to outperform two niching genetic algorithms - sequential niching [BBM93] and deterministic crowding [Mah95a] - on a set of benchmark multimodal function optimization problems [BEvdB03].

3.3.3 Parallel Vector-based PSO

Schoeman and Engelbrecht proposed a different niching approach in [SE04], and implemented it in the Vector-based PSO (VPSO). Their approach considered the two vector components of the velocity update formula in Equation (2.5), which point respectively towards the particle's best position and the neighborhood best. When this two vectors point roughly towards the same direction, the particle will modify its trajectory towards the neighborhood best, otherwise, it will probably head for another optimal solution. Niches are identified according to this criterion. In fact, the dot product of the two vectors is calculated:

$$\lambda_i = \mathbf{v}_{pi} \cdot \mathbf{v}_{gi} = \|\mathbf{v}_{pi}\| \|\mathbf{v}_{gi}\| \cos\theta_i, \tag{3.14}$$

where the two vectors are:

$$\mathbf{v}_{pi} = \mathbf{p}_i - \mathbf{x}_i \tag{3.15}$$

$$\mathbf{v}_{gi} = \mathbf{p}_g - \mathbf{x}_i. \tag{3.16}$$

The dot product λ_i will be positive if the two vectors point in the same direction and negative otherwise. Once λ_i is determined for each particle i in the swarm,

Algorithm 3.4 Parallel Vector-based PSO pseudocode.

```
1: procedure PVPSO
2:     Initialize the particle swarm.
3:     repeat
4:         Set g to the index i of the unprocessed particle with the best p_i.
5:         Calculate λ_i for each particle.
6:         Determine the niche radius R_j.
7:         Process all particles whose distance from gbest lies within R_j and put
       them into the niche S_j.
8:         if the niche S_j contains less than 3 particles then
9:             Spawn new particles in the niche.
10:        end if
11:    until each particle has been processed
12:    for m steps do
13:        for k steps do
14:            Perform the particle swarm update.
15:        end for
16:        Update λ_i and R_j.
17:        if the distance between the gbests of two niches is smaller than ε then
18:            Merge the two niches.
19:        end if
20:    end for
21: end procedure
```

the *niche radius* can be defined as the distance from the neighborhood best of the closest particle with a negative λ_i:

$$R_j = \min_{i:\lambda_i<0} \|\mathbf{x}_g - \mathbf{x}_i\|. \tag{3.17}$$

The original VPSO algorithm identified and exploited niches in a sequential way, starting from the global best and then repeating the procedure for the particles outside its niche. In [SE05], the same authors developed the Parallel Vector-based PSO (PVPSO), a more efficient version, based on the same principles, but which followed a parallel approach. In this case, the different niches are identified and maintained in parallel, with the introduction of a special procedure which can merge

Algorithm 3.5 The procedure for determining the species seeds in SPSO.

```
1: function SEEDS(L: the set of particles in the swarm)
2:     Sort particles in L based on decreasing fitness values.
3:     Initialize the set of seeds: S = ∅.
4:     for each particle i in L do
5:         Let found = false.
6:         for each seed j in S do
7:             if d(x_i, x_j) < σ then
8:                 Let found = true.
9:                 Break.
10:            end if
11:        end for
12:        if not found then
13:            Add particle i to S.
14:        end if
15:    end for
16:    return the set of seeds S.
17: end function
```

two niches when they become closer than a specified threshold ϵ. The pseudocode for the PVPSO algorithm is given in Algorithm 3.4.

The vector-based approach has the appealing property of identifying niches by using operations on vectors which are inherent to the particle swarm algorithm. Thus it provides an elegant way to build a particle swam for multimodal function optimization. However, when it was tested on common benchmark functions, the results showed that its performances were lower than those of other approaches, such as the NichePSO.

3.3.4 Species-based PSO

Another niching version of the particle swarm algorithm, the Species-based PSO (SPSO) was proposed by Li [Li04]. The approach was the adaptation to the particle swarm of the Species Conserving Genetic Algorithm [LBPC02], which we discussed early, and of whom Li itself had been among the proponents. In particular, SPSO adopted the same procedure for determining the species seeds, which identify the

Algorithm 3.6 Species-based PSO pseudocode.

```
1: procedure SPSO
2:     Initialize the particle swarm.
3:     repeat
4:         Evaluate the particles in the swarm.
5:         Execute function Seeds.
6:         Assign each particle's nbest to the closest seed.
7:         Perform a standard particle swarm step.
8:     until the termination criteria are met.
9: end procedure
```

niches in the population. This procedure, which we report in Algorithm 3.5, is in fact essentially analogous to the clearing procedure proposed by Petrowski (see Section 3.2.4).

Once the species seeds have been identified, all the other particles are assigned to the niche formed by the closest seed, and the neighborhood structure is adapted to reflect the division in niches. In fact, each species seed would serve as the *nbest* for all the other particles in its niche. A description of the SPSO algorithm is reported in Algorithm 3.6.

The SPSO niching approach can dynamically identify the number of niches in the fitness landscape, and also proved to be suitable for the application on dynamic environments [PL04]. However, it still requires a radius parameter σ to determine the extension of the niches. Moreover, it implicitly assumes that all the niches have roughly the same extension.

3.3.5 Adaptive Niching PSO

In [BL06], Bird and Li developed a new algorithm which could adaptively determine the main niching parameters: the Adaptive Niching PSO (ANPSO).

The first step of the algorithm calculates r, the average distance between each particle and its closest neighbor:

$$r = \frac{\sum_{i=1}^{N} \min_{j \neq i} \|\mathbf{x}_i - \mathbf{x}_j\|}{N}. \tag{3.18}$$

This value is then used to determine the formation of niches. In fact, ANPSO

Algorithm 3.7 Pseudocode for the niche formation procedure in ANPSO.

```
1: function NICHES(s: array of N × N of integers ∈ [0, 4])
2:     Determine r using Equation (3.18).
3:     Create an undirected graph 𝒢 with a node for each particle and no edges.
4:     for i = 1 → N − 1 do
5:         for j = i + 1 → N do
6:             if ‖x_i − x_j‖ < r then
7:                 s_ij = min{s_ij + 1; 4}.
8:                 if s_ij ≤ 2 then
9:                     Create an edge in 𝒢 from x_i to x_j.
10:                end if
11:            else
12:                s_ij = max{s_ij − 1; 0}.
13:            end if
14:        end for
15:    end for
16:    Create the niches from the connected subgraphs in 𝒢.
17:    return the niches.
18: end function
```

keeps track of the minimum distance between particles over a number of steps. At each iteration, the graph $\mathcal{G}$ with particles as nodes is considered, and an edge is added between every pair of particles which have been closer than r in the last 2 steps. Niches are formed from the connected subgraphs of $\mathcal{G}$, whilst all the particles which end up with no edges remain outside any niches. The procedure for the formation of niches is described in Algorithm 3.7. As it can be seen from Equation (3.18), the computational cost of the niche formation procedure is $O(N^2)$ with respect to distance calculations, which is quite expensive in comparison to other techniques.

ANPSO executes a particle swarm simulation with constriction factor, but redefines the neighborhood topology at each step (see Algorithm 3.8). In fact, once it determines the niches with the procedure in Algorithm 3.7, it uses a *gbest* topology for each niche, and a von Neumann topology for non-niched particles. In this way, the particles which have formed a niche will tend to perform a local search around an optimum, whilst the others will continue searching the whole space.

Algorithm 3.8 ANPSO pseudocode.

```
1: procedure ANPSO
2:     Initialize the particle swarm.
3:     repeat
4:         Evaluate the particles in the swarm.
5:         Execute function NICHES.
6:         for each particle i do
7:             if i belongs to a niche S_j then
8:                 Assign i to a gbest topology on S_j.
9:             else
10:                Assign i to a global von Neumann topology.
11:            end if
12:        end for
13:        Perform a standard particle swarm step.
14:    until the termination criteria are met.
15: end procedure
```

ANPSO provides for a much more flexible technique when compared to others, e.g. SPSO, which require critical, problem-dependent parameters, such as the niche radius. However, this flexibility is paid with the rather higher computational cost of the niche formation procedure. Although this added cost can seriously slow down the algorithm, it should be considered that other approaches often require multiple runs for the fine-tuning of the parameters, whilst ANPSO could be executed just once. Moreover, for most complex problems, the cost of function evaluations is much higher than that of distance calculations involved in the niching procedure, rendering its overhead less significant.

Chapter 4

Clustering Particles

In this chapter we will introduce a new approach to niching for the particle swarm, which is based on clustering. We will explain the basic concept behind the approach, present the clustering algorithm we will use, the k-means, and discuss the first implementation of the kPSO algorithm. Then we will follow the development of the new algorithm, improving its performance and comparing it to other niching approaches for PSO.

4.1 The basic concept

The Particle Swarm Optimization algorithm has many points in common with evolutionary algorithms. Certainly enough to allow for the application of the concept of niching, originated in the evolutionary framework. However, some peculiarities of PSO must be taken into account. In PSO there is no selection mechanism: all interactions among particles take place along the neighborhood structure. Therefore, a niching mechanism should operate directly or indirectly on it, by modifying neighborhood relationships among particles. Another aspect by which PSO differs from the evolutionary approach consists of the fact that particles keep a memory of their previous best positions. This information can definitely be exploited in the discovery of niches.

We discussed in Section 3.3 about the *implicit* niching capabilities which the PSO enjoys. Though they are not sufficient to provide for an effective method for the location of multiple optima, they are a good starting point to build one. As stated by Kennedy and Eberhart:

> After a few iterations, particles in the particle swarm are seen to

> cluster in one or several regions of the search space. These clusters indicate the presence of optima, where individuals' relatively good performances have caused them to attract their neighbors, who in moving towards the optimal regions improved their own performances, attracting *their* neighbors, and so on. ([KE01])

In fact, the tendency of particles to group near the optima of the function is balanced in the end by the social influence of the particle which has found the best position. Depending on the specific neighborhood topology, this influence will extend to the other particles in the swarm at different rates. However, since the graph representing the neighborhood relationship is in general a connected graph, eventually all particles will tend to be attracted towards a single position.

The basic idea behind our approach is to dynamically adapt the neighborhood structure in order to form different niches in the swarm. This is accomplished by applying a standard clustering algorithm to identify niches and then restricting the neighborhood of each particle to the other particles in the same cluster. In this way each cluster of particles tends to perform a local search in the function domain and to locate a different optimum.

4.2 Stereotyping

The use of clustering on the particles of a swarm is not new. In [Ken00], Kennedy introduced the concept of *stereotyping* and discussed its impact on the performance of the PSO algorithm. A stereotype is an individual or the idealization of an individual which represents the *norm* of a group. The algorithm presented by Kennedy used clustering to identify different groups in the swarm and defined the centroid of each cluster as the stereotype which the particles in that cluster would look at.

Kennedy's study was related to the social metaphor underlying the particle swarm. In the standard PSO, particles are attracted towards their personal best position and the best position among their neighbors. This attraction simulated the tendency to follow the best individual of a group. The application of stereotyping shifted this attraction from the best individual to a statistical prototype (the centroid of each cluster).

Stereotypes were identified by a cluster analysis on the particles' previous best positions. In particular, Kennedy used a version of the k-means clustering algorithm,

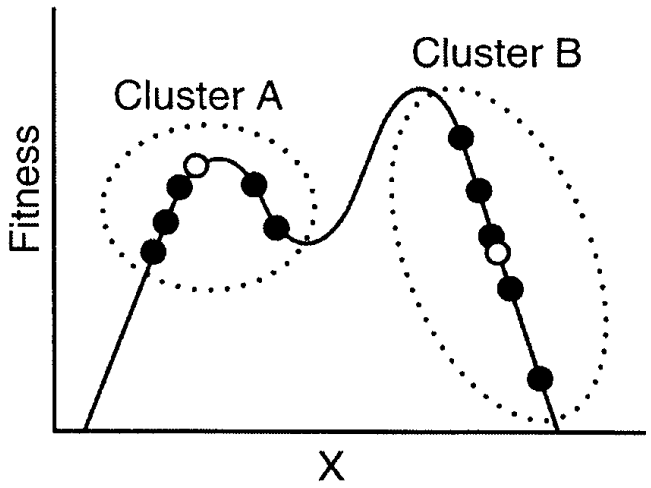


Figure 4.1: Cluster A's center (white circle performs better than any of the members of the cluster, while Cluster B's center performs better than some, and worse than others. (Reproduced from [Ken00] – Figure 1)

which we will describe in Section 4.3. The experiments on stereotyping were aimed at studying the swarm behavior when the cluster center was used in place of the individual (i) best or of the neighborhood (g) best in the velocity update formula. Thus, four variation were tried:

1. The formula used i's and g's individual previous best. (the standard swarm)
2. The individual best term was replaced with i's cluster's center.
3. The neighborhood best term was replaced with g's cluster's center.
4. Both best terms were replaced with cluster centers.

The performance of this variations of the particle swarm algorithm were tested on a standard set of benchmark functions. Results showed that the 2^{nd} version was the only one with a clear advantage over the standard swarm. That is, substituting the individual previous best with its cluster's center gave it some advantage, while substituting the neighborhood best with the respective centroid did not. A possible explanation is that using a stereotype in place of an individual is *in average* a quite good choice, whilst substituting the *best* individual with the stereotype will in general lower its fitness (see 4.1).

In the end, the use of clustering for a stereotyping approach brought some useful insight in the study of the particle swarm. However, its goals were completely

unrelated to niching. In the following we will introduce a new variation of the particle swarm technique which uses a very similar clustering procedure, but focuses on the identification of niches.

4.3 k-means

In order to implement our clustering approach to niching, we chose to use the k-means algorithm, which is probably the best-known partitional clustering algorithm [DH73, XW05]. k-means is a very simple algorithm and, as we have just seen, it had already been applied by Kennedy to cluster particles in a swarm in his research on stereotyping.

In the following we will briefly describe the k-means algorithm in its application to the particles of the swarm. The algorithm works by partitioning the search space according to k randomly initialized points $\mathbf{m}_1, \ldots, \mathbf{m}_k$ – the *seeds*. Each particle i in the swarm is assigned to the cluster $C_{k'}$ associated with the seed which is closest to its previous best position $\mathbf{p}_i$:

$$k' = \arg \min_{1 \leq j \leq k} \|\mathbf{p}_i - \mathbf{m}_j\|. \tag{4.1}$$

Then the seeds are recalculated as the *means* of the particles' *pbest*s in each cluster:

$$\mathbf{m}_j = \frac{1}{N_j} \sum_{\mathbf{p} \in C_j} \mathbf{p}, \tag{4.2}$$

where N_j is the number of particles in C_j. The partitioning is updated until no change occurs. In Algorithm 4.1 the k-means procedure is reported in detail.

The time complexity of the k-means algorithm is $O(N \cdot k)$, and since usually the number of clusters k is much lower than the number of particles N, it is rather quick. The clusters resulting from the application of the algorithm are typically compact and hyperspherical, since k-means implicitly tends to minimize the squared error (or scattering) J:

$$J = \sum_j \sigma_j, \tag{4.3}$$

where σ_j is the variance of the cluster C_j:

Algorithm 4.1 k-means algorithm pseudocode.

```
1: procedure k-MEANS
2:     Initialize the seeds m_1, ..., m_k.
3:     repeat
4:         for each particle i do
5:             Find the nearest seed m_k' (Equation (4.1)).
6:             Assign i to the cluster C_k'.
7:         end for
8:         for each cluster C_j do
9:             Recalculate the new mean (Equation (4.2)).
10:        end for
11:    until no change occurs
12: end procedure
```

$$\sigma_j^2 = \frac{1}{N_j - 1} \sum_{\mathbf{p} \in C_j} \|\mathbf{p} - \mathbf{m}_j\|^2. \tag{4.4}$$

The main issues with the k-means algorithm are twofold:

1. It is a heuristic algorithm, which does not guarantee to converge on an optimal solution, and, above all, strongly depends on the initial partition. For our goals it is probably not a great issue not to have a perfect clustering, since we need it just to roughly group particles in different niches. However, the dependency on the initial seeds can be a problem. In fact, it is most commonly overcome by repeating the k-means procedure a number r_k of times with different random seeds. In the end the clustering which minimizes J (Equation (4.3)) is chosen. We chose to follow this approach, which, however, brings an additional computational cost.

2. The number of clusters k is to be fixed *a priori*. In many applications this can be a serious issue, as there may be no information about the natural number of clusters in the data to be analyzed. In our case, the number of clusters should be higher than the number of expected optima in the function under consideration. Thus it is a parameter for our method which is much dependent on the problem. In the following we will discuss in detail how different values

for k influence our clustering–based PSO and how it can be modified to adapt the value of k during the simulation.

4.4 k-means Particle Swarm Optimization

The algorithm we developed is the k-means Particle Swarm Optimization (kPSO), the first version of which was introduced in [PS06]. It provides a basic implementation of our clustering approach to niching. In particular, we employed the standard k-means algorithm described in the previous section to cluster particles according to their *pbest*, the previous best position.

kPSO uses the clusters of particles to modify the neighborhood topology so that each particle can communicate only with particles in the same cluster. Therefore, the swarm turns in a collection of *sub-swarms* which tend to explore different regions of the search space. Since our goal is that the sub-swarms perform a rather local search, each of them uses a *gbest* topology, with all the particles in a cluster connected to each other.

Clustering is performed after the swarm random initialization and then repeated at regular intervals during the swarm simulation. Between two clustering applications the swarm (or more precisely, the sub-swarms corresponding to the various clusters) follows its (their) normal dynamics. In fact, the multiple applications of the clustering algorithm are meant to keep track of the swarm dynamics: particles in different clusters at early stages of the simulation can end up in the same cluster as they move towards the same local optimum or, in contrast, a single cluster can be split into two as some of its particles fly towards a different optimum.

A relevant difference between our approach and other niching PSO techniques is the fact that the clustering algorithm is not applied at each step of the swarm simulation, but only every c steps. The rationale behind this choice relies on the natural tendency of the swarm to cluster around the function optima. The application of a clustering procedure should just enhance this natural tendency and, above all, maintain the clusters over time by blocking communication between particles in different clusters. However, if the algorithm re-identified the sub-swarms at each step, the particles would not have time to follow their natural dynamics. Therefore, we let them evolve for some steps following the standard PSO dynamics.

In the following, we will discuss in detail the consequences of this approach, and

Algorithm 4.2 The procedure to identify niches in the kPSO algorithm.

```
1: procedure IdentifyNiches
2:     Cluster particles' pbests with the k-means algorithm.
3:     Calculate the average number of particles per cluster, N_avg.
4:     Set N_u = 0.
5:     for each cluster C_j do
6:         if N_j > N_avg then
7:             Remove the N_j − N_avg worst particles from C_j.
8:             Add N_j − N_avg to N_u.
9:         end if
10:        Adapt the neighborhood structure for the particles in C_j.
11:    end for
12:    Reinitialize the N_u un-niched particles.
13: end procedure
```

set up some experiments to determine the actual effect it has on the performance of the algorithm. In the mean time, it should be noted that, as a side effect, this choice has the obvious advantage of introducing a smaller computational overhead, since the clustering procedure is applied only $\sim T/c$ times (T being the total number of simulation steps).

After performing the clustering, a cutting procedure is applied: we measure the average number of particles per cluster N_{avg} and proceed by removing the exceeding particles from the clusters which are bigger than the average. To this end, we keep the particles in each cluster C_j sorted by their previous best position fitness, so that we can remove the worst $N_j - N_{avg}$ particles of the cluster.

The goal of the cutting procedure is to avoid the formation of overcrowded niches, which would end up in a waste of computational power, as too many particles would explore the search space around a single optimum. Instead, the N_u exceeding particles removed from the overcrowded clusters can be randomly reinitialized, in order to explore new areas.

Niches are modeled after the reduced clusters, as described in Algorithm 4.2. Each particle's neighborhood is set to the ensemble of particles in the same cluster. Thus we will have k (the number of clusters) niches whose particles are fully–connected (see Figure 4.2), realizing a *gbest* topology in each niche.

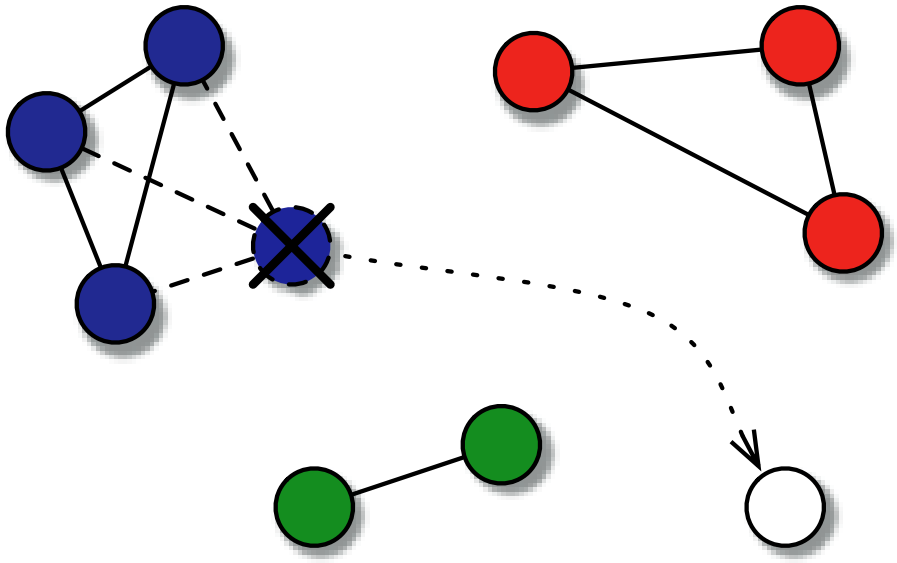


Figure 4.2: Particles are linked to all the other particles in the same cluster. Clusters with a number of particles greater than the average are reduced and the exceeding particles reinitialized (see the particle in white).

The remaining particles, which were removed from the overcrowded clusters and reinitialized, are used to explore the search space for possible new interesting areas. In the first implementation of the algorithm, these *un-niched* particles will have no connections. Thus they will perform the kind of non-deterministic hill-climbing associated with the *cognition only* model [Ken97] (see also Section 2.4.3).

Once the neighborhood structure has been set both for the niched and the un-niched particles, the algorithm performs a fixed number c of steps of the constriction-factor PSO (see Section 2.3.1). In this phase there are only two aspects in which kPSO differs from the standard algorithm:

1. The particles assigned to a niche C_j will have their velocities clamped to $V_{max} = 2\sigma_j$ (see Equation (4.4)), which is proportional to the width of the cluster.

2. The un-niched particles will behave as in a *cognition only* model.

After c steps, the clustering and cutting procedures are repeated. The pseudo-code for kPSO is reported in Algorithm 4.3.

The application of the kPSO algorithm requires to set a few additional parameters with respect to the standard PSO. We already discussed about the first

Algorithm 4.3 kPSO algorithm pseudocode.

```
1: procedure kPSO
2:     Initialize particles with random positions and velocities.
3:     Set particles' pbests to their current positions.
4:     Calculate particles' fitness and set gbest.
5:     for step t = 1 → T do
6:         if t mod c = 0 then                          ▷ Every c steps.
7:             Execute the procedure IdentifyNiches.
8:         end if
9:         Update particles' velocities.          ▷ Perform the standard PSO step.
10:        Update particles' positions.
11:        Recalculate particles' fitness.
12:        Update particles' and neighborhood best positions.
13:    end for
14: end procedure
```

one, c, the number of PSO steps between two clustering applications.

The other additional parameters are strongly tied to the specific clustering algorithm chosen, k-means. Since the k-means algorithm depends on the initial assignment of the seeds, it must be repeated a number r_k of times for a single clustering application. The optimal clustering is chosen by selecting the one with minimal scattering J (see Section 4.3).

The second parameter for the k-means algorithm is k, the number of clusters, which is also the most problematic, as it is strongly dependent on the number of optima of the function under consideration. When this number is known, k can be set to a value which is slightly larger than it. In this way, the algorithm will maintain a sufficient number of clusters to be able to identify all the optima, while possibly forming spurious clusters, that is clusters not corresponding to any optima. In the following, we will set up some experiments which will better explain the dynamics of kPSO.

4.4.1 First experiments

The first experiments on the kPSO algorithm have two main goals:

1. to provide a first confirmation that the clustering approach is indeed feasible;
2. to help setting the new parameters and understanding their influence on the performance of the algorithm.

Therefore, we chose to start the study using multimodal functions which have been built *ad hoc* for the purpose. In fact, we will consider two functions which are both the sum of multiple two-dimensional gaussians, each in the form:

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{(x-\mu_x)^2+(y-\mu_y)^2}{2\sigma^2}}, \tag{4.5}$$

thus with mean (μ_x, μ_y) and variance $\sigma\mathcal{I}_2$ (where $\mathcal{I}_2$ is the identity matrix in $\mathbb{R}^2$). The resulting functions will have an optimum for each gaussian, and its position will be the mean of the gaussian. The two functions we considered for the tests differ for the number of optima:

1. The first function, $G1$, is the sum of 5 gaussians, all of them with $\sigma = 0.4$, and with the following means:

$$G1 : \{(0,0),(1,1),(-1,1),(-1,-1),(1,-1)\}. \tag{4.6}$$

   $G1$ was studied in the range $(x,y) \in [-2,2] \times [-2,2]$, as plotted in Figure 4.3.

2. The second one, $G2$, is the sum of 25 gaussians. The variance is still $\sigma = 0.4$ for all of them, while their means are:

$$\begin{aligned} G2 \;:\; & \{(0,0),(1,1),(-1,1),(-1,-1),(1,-1),\\ & (2,0),(2,2),(0,2),(-2,-2),(-2,0),\\ & (-2,-2),(0,-2),(2,-2),(3,1),(3,3),\\ & (1,3),(-1,3),(-3,3),(-3,1),(-3,-1),\\ & (-3,-3),(-1,-3),(1,-3),(3,-3),(3,-1)\}. \end{aligned} \tag{4.7}$$

   $G2$ was studied in the range $(x,y) \in [-4,4] \times [-4,4]$, as plotted in Figure 4.4.

In all the tests reported in this section, we measured the performance of the kPSO algorithm varying the main parameters of the algorithm. Regarding those parameters which are in common with the standard constriction-factor PSO, we used

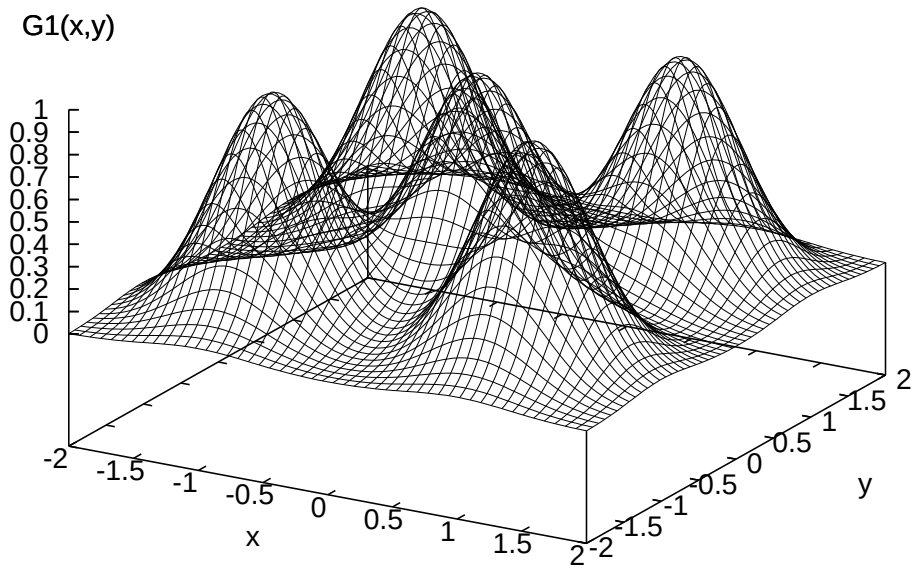


Figure 4.3: Function $G1$.

the common values of $p_{incr} = g_{incr} = 2.05$ and $\chi \simeq 0.730$. The number of particles obviously varied for the two functions: we used 30 particles in the experiments on function $G1$ and 150 in the experiments on function $G2$.

We repeated each experiment 100 times, and calculated the average values of two performance measures:

1. *Optima (%)*: The percentage of optima the algorithm can find with respect to the total number of optima of the function under consideration, within a maximum of $T = 1000$ iterations. An optimum is considered to be found when the best position of a particle in a niche is within a distance of $d_\epsilon = 0.1$ from the optimal position and the function value is within $\epsilon = 0.001$ from the optimal value. The first threshold, d_ϵ, is needed just to distinguish between different optima, so it can be set to a value roughly smaller than half the minimum distance between two optima. Instead, the second one, ϵ, actually defines the degree of refinement of the solutions.

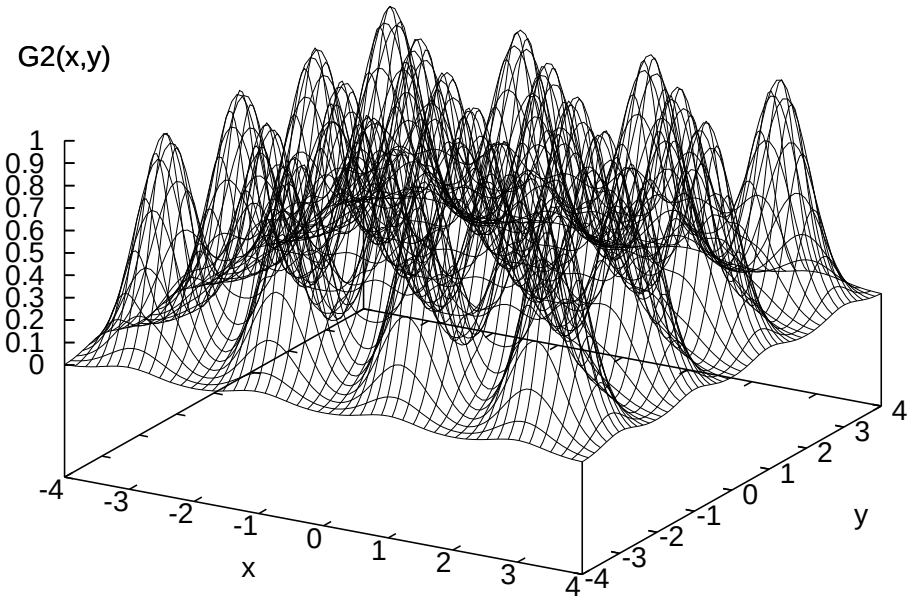


Figure 4.4: Function $G2$.

2. *No. of steps*: The number of steps required to find all the optima of the function under consideration. Actually, we check that the algorithm also maintains all the optima for at least 10 iterations after they have been found.

In order to study the influence of the new parameters introduced in the algorithm by the niching strategy, our methodology devised a first exploratory phase which had the aim to determine a good choice of values for each of the parameters on the two functions $G1$ and $G2$. Then a series of tests were conducted in which we let one of the parameters vary in a wide range while keeping the others fixed at their optimal values. In Table 4.1 we report the parameters values which have been selected in the exploratory phase and then used in the successive tests as fixed values for the parameters which were not the subject of the tests.

Table 4.1: Selected parameters for the experiments on functions $G1$ and $G2$. In each series of tests, one of the parameters c, k, and r_k are varied, while the other two are kept fixed at their optimal values.

Function	Optima	Particles	c	k	r_k
$G1$	5	30	6	7	10
$G2$	25	150	25	40	10

c: the number of steps before clustering

The first parameter under study is c, the number of steps between two applications of the clustering procedure. We run the kPSO algorithm on the two test functions using values for c from 1 to 50.

In Figure 4.5 we plotted the results obtained on the simpler function $G1$. The first thing to note is that the algorithm succeeded in locating all the optima of the function with every value of c larger than 1. When $c = 1$, that is, when the clustering is repeated at every step, the algorithm could find on average only 98% of the optima. The average number of steps required to find the optima, instead, was very low when the clustering was repeated every a few iterations, roughly between 3 and 10, then it grew almost linearly with c. A possible explanation is that the swarm needed a few steps to localize the optima, but then it remained stuck until a new clustering was applied. After that, it could actually find new optima.

The behavior of kPSO when applied on function $G2$ followed a similar pattern (see Figure 4.6). On this more complex function, the algorithm needed more steps between two clustering applications. In fact, when $c = 1$ it could not find more than $\sim 88\%$ of the optima, for greater values the percentage went up to $\sim 99\%$, then after $c \simeq 20$ it stabilized on $\sim 99.9\%$. The average number of steps had its minimum when $20 \lesssim c \lesssim 30$, then it started growing in a similar way to the previous case.

In the end, we concluded that there is a minimal threshold for the c parameter in order for the kPSO algorithm to work at its full potential regarding the number of optima it can find. Conversely, setting c to a too high value has no negative influence on the percentage of optima, but deteriorates its performance regarding the number of required steps. Thus, we selected as the optimal values for c on the functions $G1$ and $G2$ respectively $c = 6$ and $c = 25$.

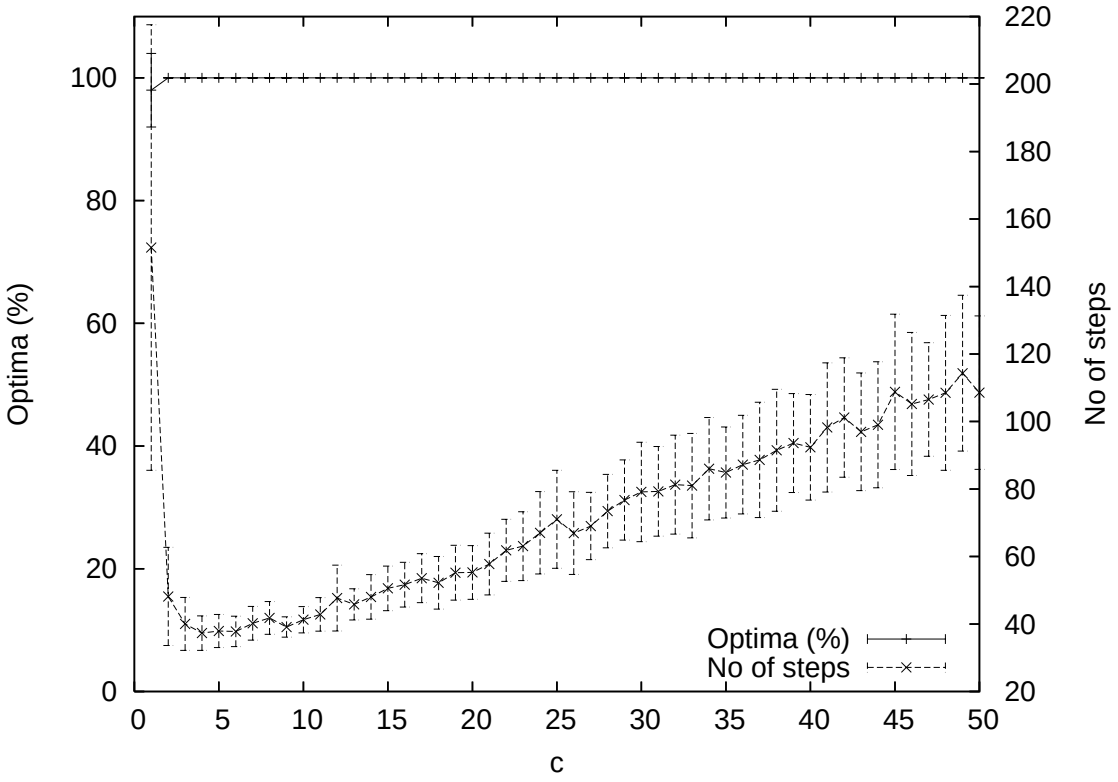


Figure 4.5: Performance of kPSO on function $G1$ with varying c. Error bars represent the standard deviation.

k: the number of clusters

The second parameter we studied was the number of clusters for the k-means algorithm, k. Also this parameter proved to be rather critical for the kPSO algorithm, since it is much dependent on the function under consideration. In a way, the issue with k is similar to that with c: as we just discussed, in fact, also the parameter c needs to be adapted to the degree of complexity of the function. However, it is really important only to set c to a value that is high enough to pass a lower threshold, whilst setting it too high has just a negative effect on performance in term of number of steps.

In setting the parameter k, one should know *a priori* the number of optima of the function. Obviously, with a value for k which is less than the number of optima, the algorithm can not identify enough niches to look for all the optima. However, if it is set to a too much higher value, the k-means algorithm can fail to identify the

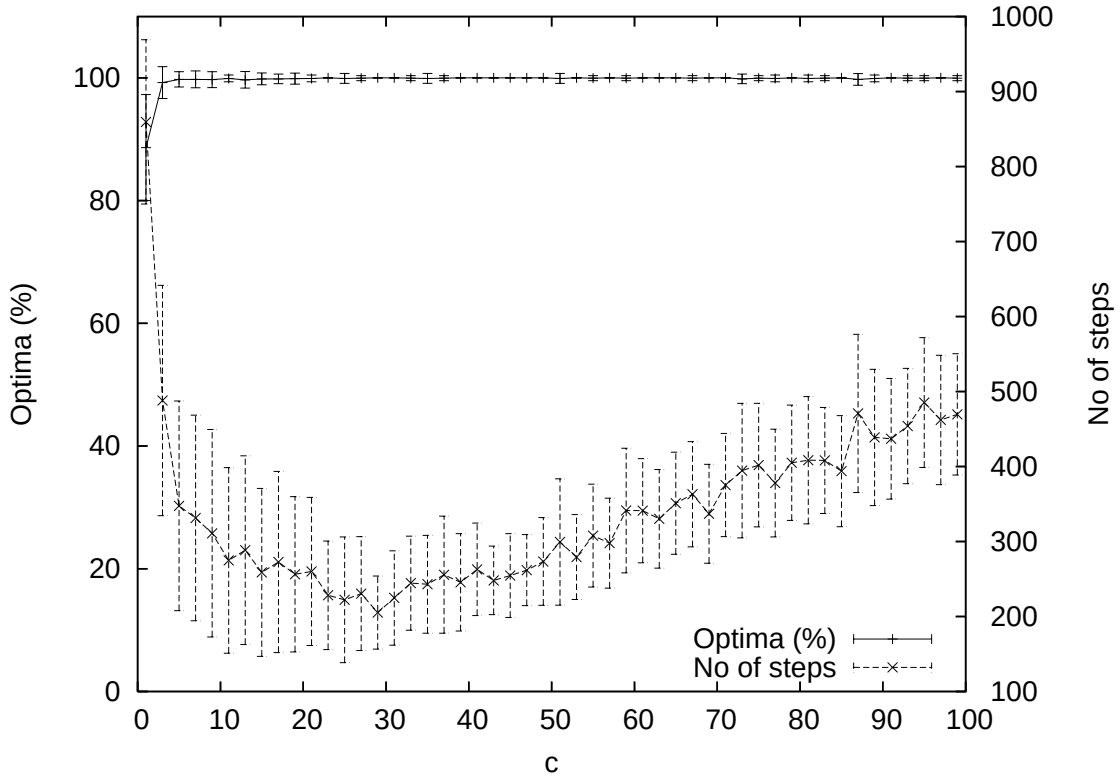


Figure 4.6: Performance of kPSO on function $G2$ with varying c. Error bars represent the standard deviation.

natural clusters the particles tend to form around the optima.

In our experiments we varied k from 1 to half the number of the particles in the swarm, thus respectively 15 for function $G1$ (see Figure 4.7) and 75 for function $G2$ (see Figure 4.8). In both cases, the algorithm started locating all the optima for values of k higher than the actual number of optima. On the other hand, with much higher values, the percentage of optima started going down again. This is probably related to the fact that increasing k too much, the algorithm ended up with clusters with just 1 or 2 particles, which could not properly behave as sub-swarms.

The second performance measure, the number of steps required to locate the optima, was strongly related to the first. In fact, it achieved the lowest values respectively with $6 \lesssim k \lesssim 9$ on function $G1$ and with $35 \lesssim k \lesssim 45$ on function $G2$. Thus we selected the two values of $k = 7$ and $k = 40$ as the optimal ones.

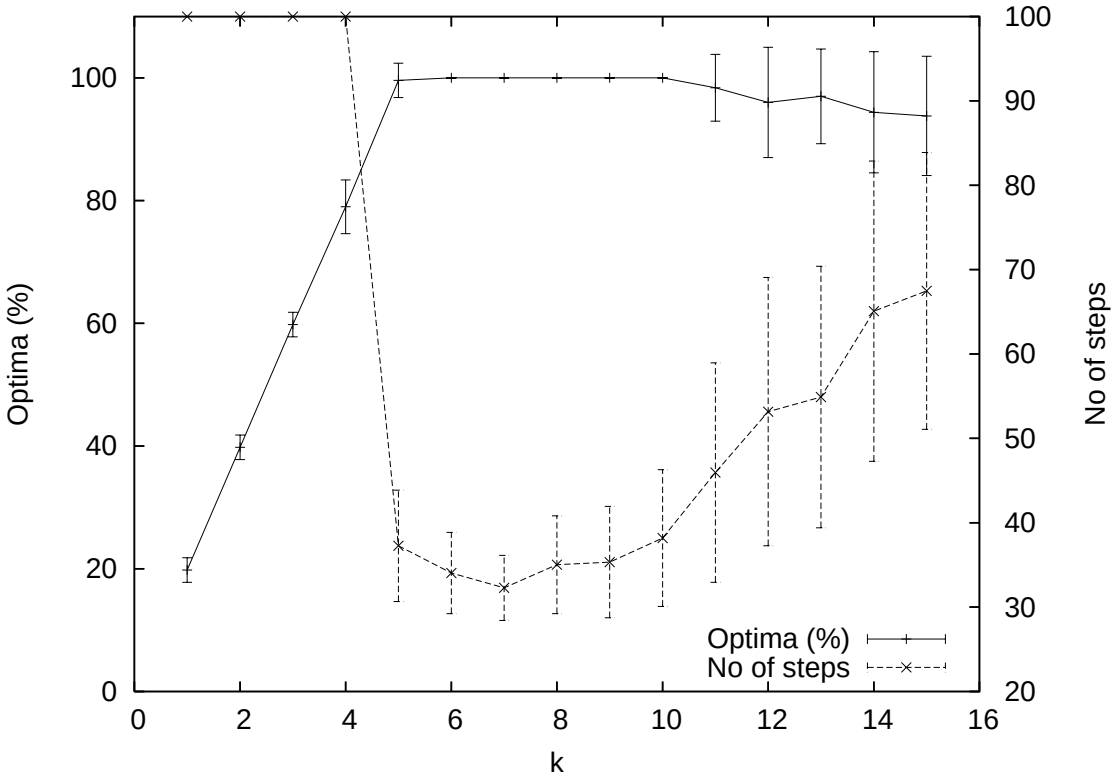


Figure 4.7: Performance of kPSO on function $G1$ with varying k. Error bars represent the standard deviation.

r_k: the number of runs of k-means

The number of runs of the k-means algorithm for each clustering application, r_k, was the most easier parameter to set. As discussed in Section 4.3, since k-means strongly depends on the initial random seeds, it must be executed multiple time to obtain a fairly good clustering.

We run the kPSO algorithm on the functions $G1$ and $G2$ with r_k varying from 1 to 50. As shown in Figure 4.9 and Figure 4.10, a number of repetitions in the order of $r_k = 10$ is sufficient in both cases to find a good clustering.

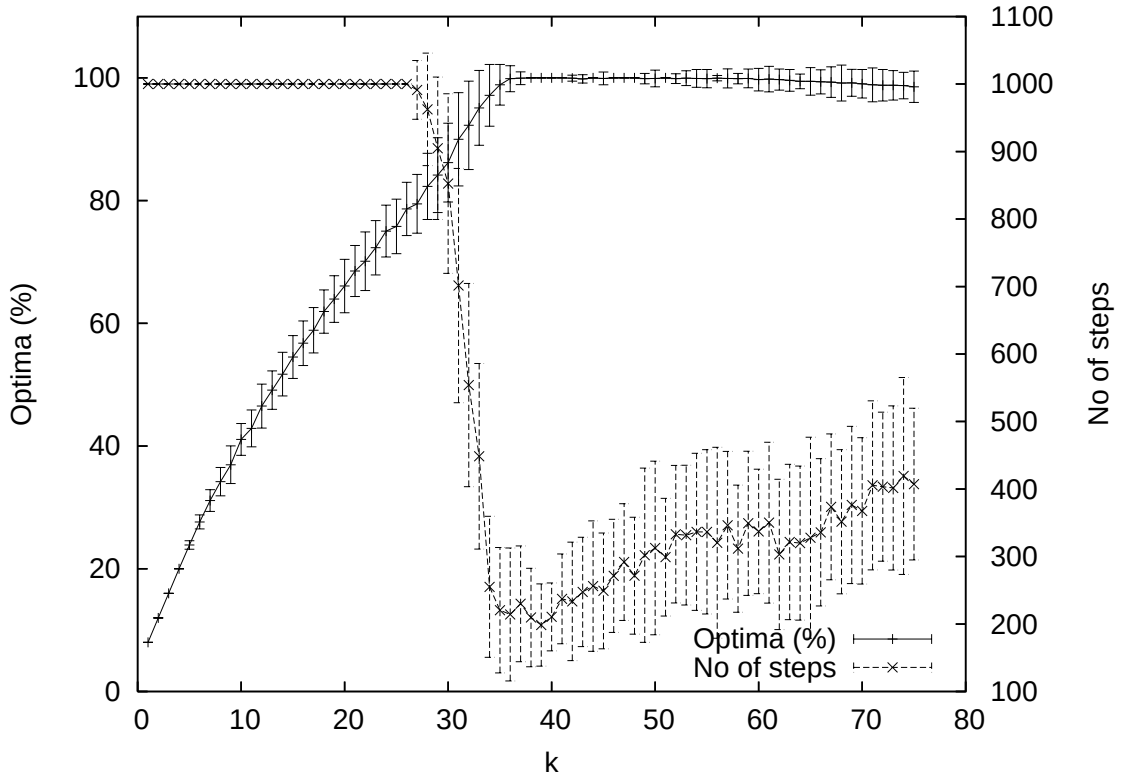


Figure 4.8: Performance of kPSO on function $G2$ with varying k. Error bars represent the standard deviation.

4.5 Comparison with other algorithms

The first experiments we presented in the previous section were mostly intended to investigate the potential of the kPSO algorithm and to gain some information about parameter settings. For this reason, the functions employed were built *ad hoc* for the task.

In this section, our goal is to provide a first comparison of our algorithm with other niching techniques for the particle swarm. Therefore, we will report and slightly extend on the analysis published in [PS06], where kPSO was introduced for the first time.

The analysis will focus on the capability of the niching algorithm to locate all the optima of the functions in a benchmark set which is widely used in literature. In this phase we are not yet considering neither the accuracy of the solutions the

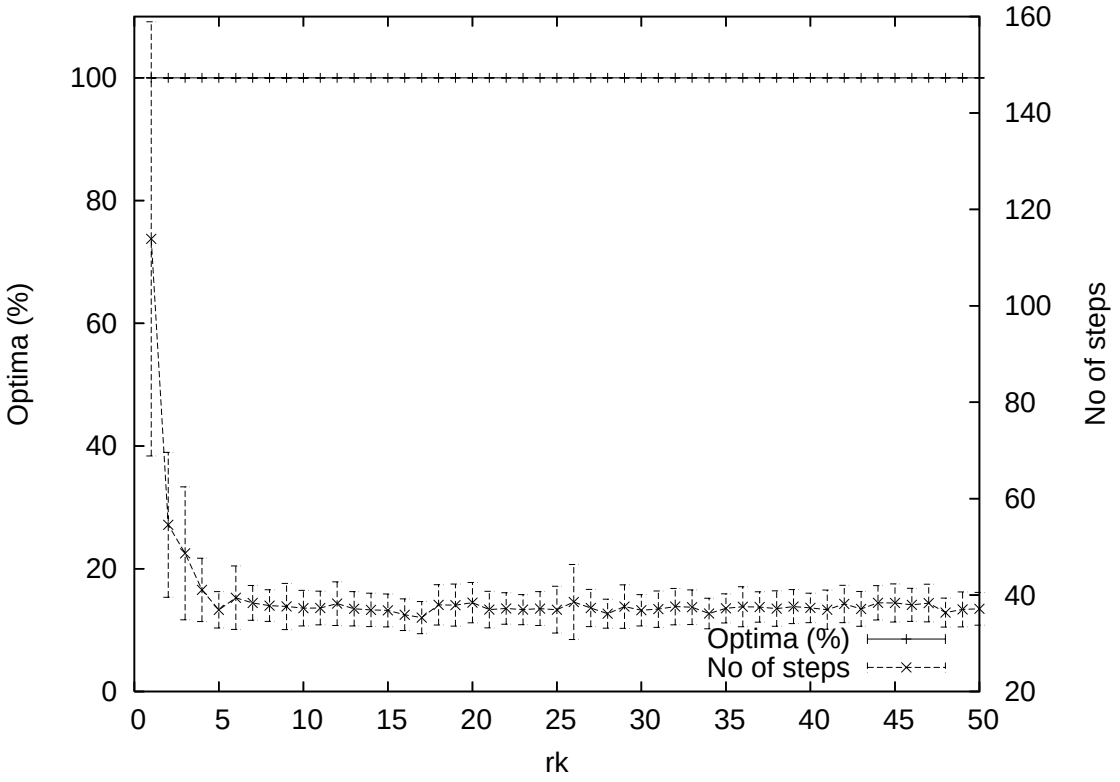


Figure 4.9: Performance of kPSO on function $G1$ with varying r_k. Error bars represent the standard deviation.

algorithm can find, nor the computational cost of the optimization.

4.5.1 Experimental setup

Our experiments reproduce the same setup reported in [SE05], in order to perform a fair comparison with the performance of the other algorithms. The benchmark set of multimodal functions include 4 one-dimensional functions and 3 two-dimensional ones.

The first 4 one-dimensional functions are defined by the following equations:

$$F1(x) = \sin^6(5\pi x). \tag{4.8}$$

$$F2(x) = e^{-2\log(2)\left(\frac{x-0.1}{0.8}\right)^2} \sin^6(5\pi x). \tag{4.9}$$

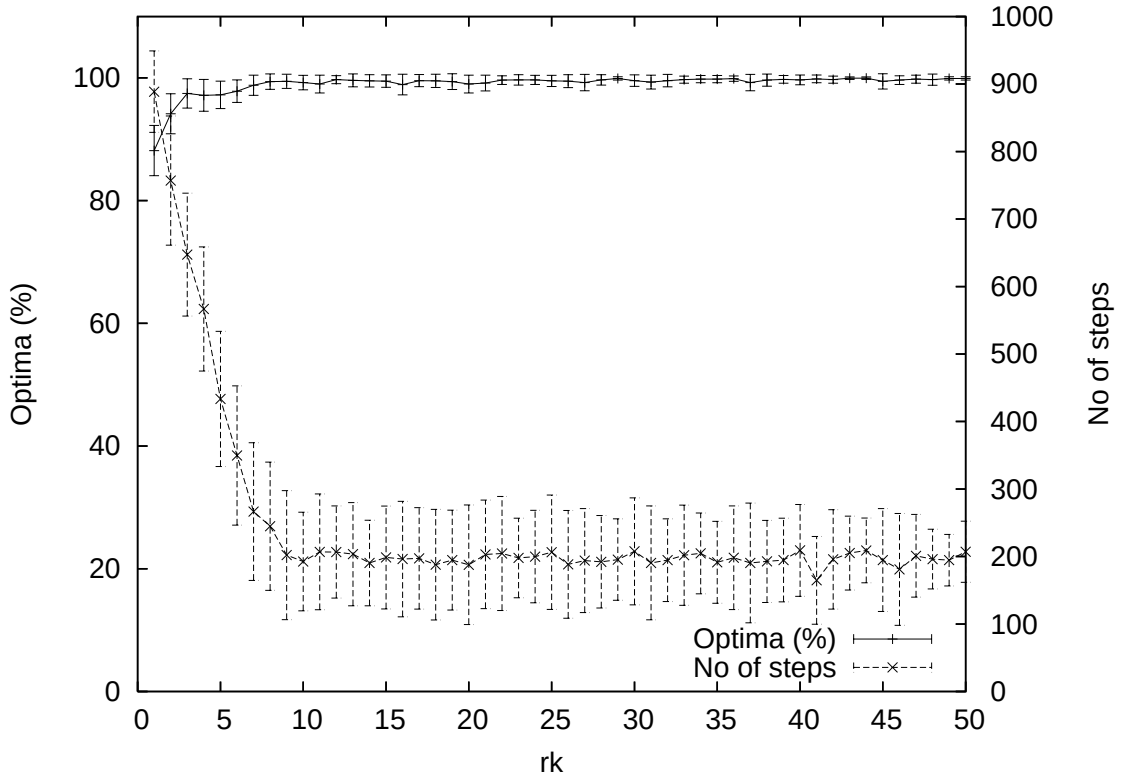


Figure 4.10: Performance of kPSO on function $G2$ with varying r_k. Error bars represent the standard deviation.

$$F3(x) = \sin^6(5\pi(x^{\frac{3}{4}} - 0.05)). \tag{4.10}$$

$$F4(x) = e^{-2\log(2)\left(\frac{x-0.08}{0.854}\right)^2} \sin^6(5\pi(x^{\frac{3}{4}} - 0.05)). \tag{4.11}$$

All of them have been studied in the range $x \in [0, 1]$, in which they all have 5 maxima. Function $F1$ is the simplest one of the set: its optima are evenly spaced and have the same height (see Figure 4.11). The successive functions start adding some degree of complexity: function $F2$ has its optima located at the same positions, but with exponentially decreasing heights (see Figure 4.12); function $F3$ has again all the 5 optima at the same height, but they are unevenly spaced (see Figure 4.13). Finally function $F4$ combines the previous two, with unevenly spaced optima at different heights (see Figure 4.14).

The remaining 3 two-dimensional functions are:

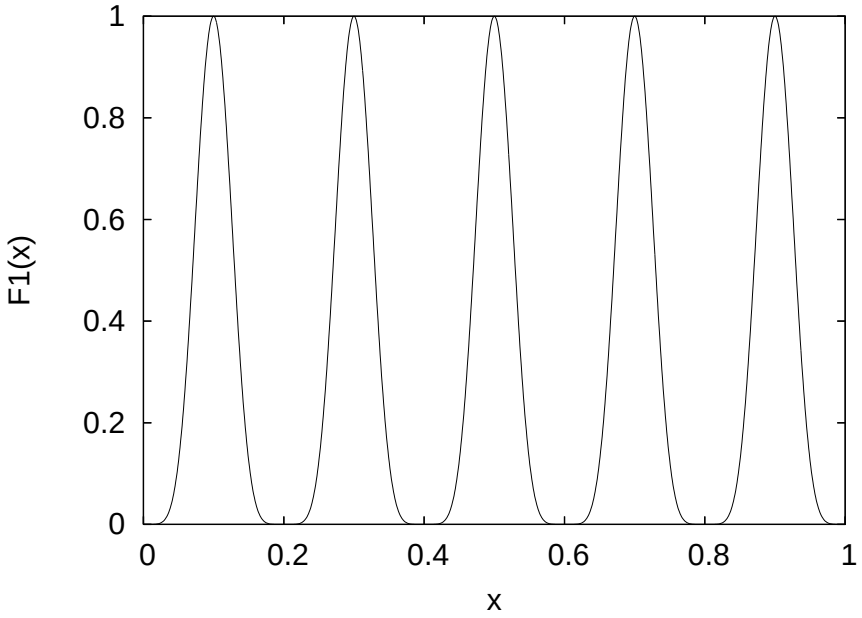


Figure 4.11: Function $F1$.

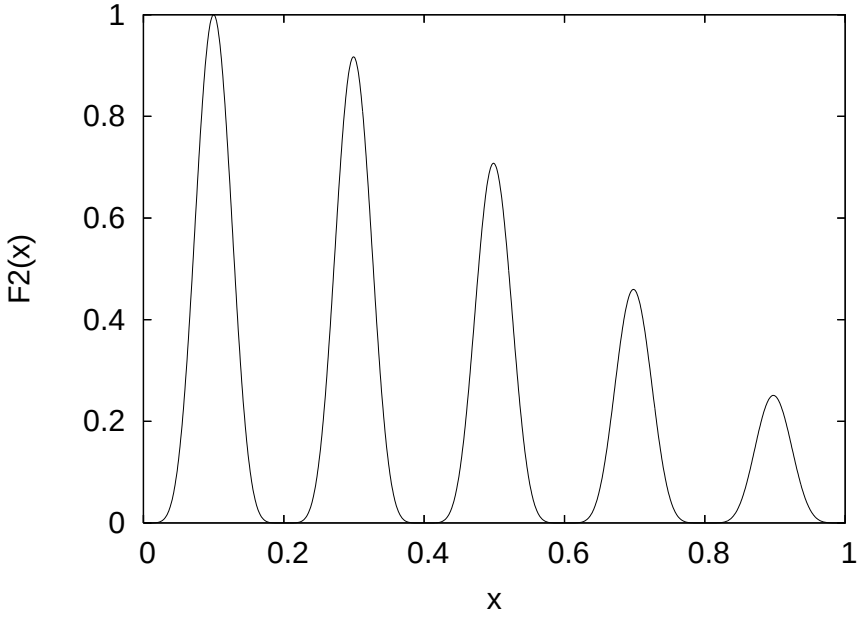


Figure 4.12: Function $F2$.

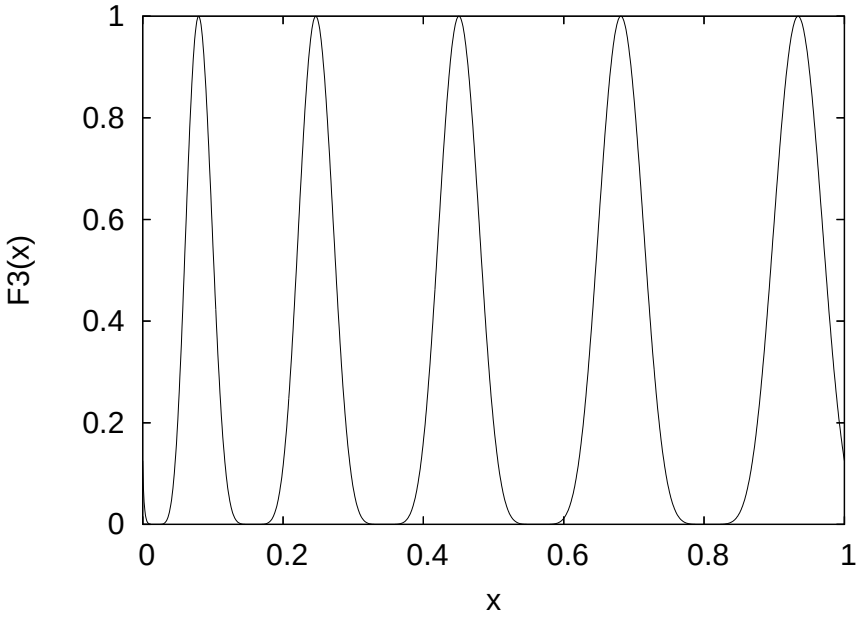


Figure 4.13: Function $F3$.

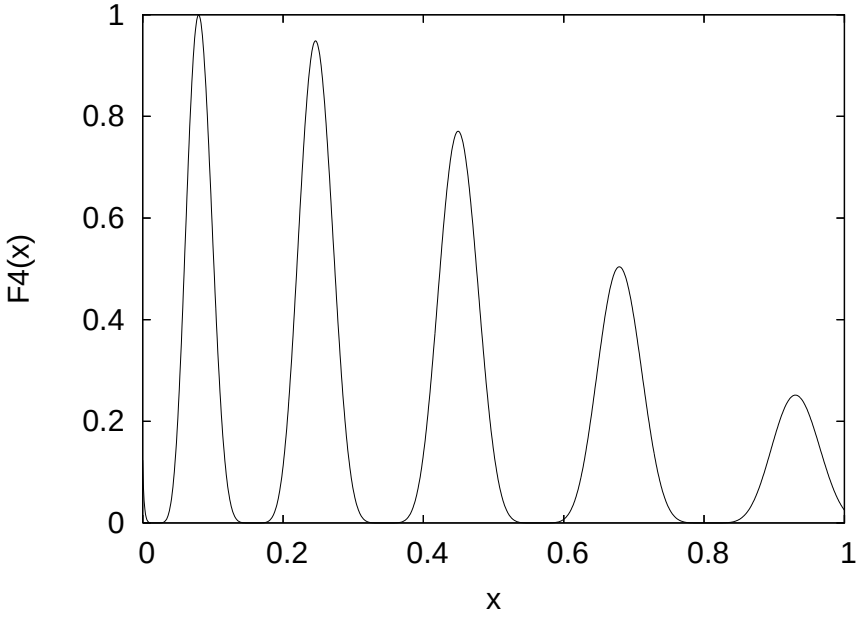


Figure 4.14: Function $F4$.

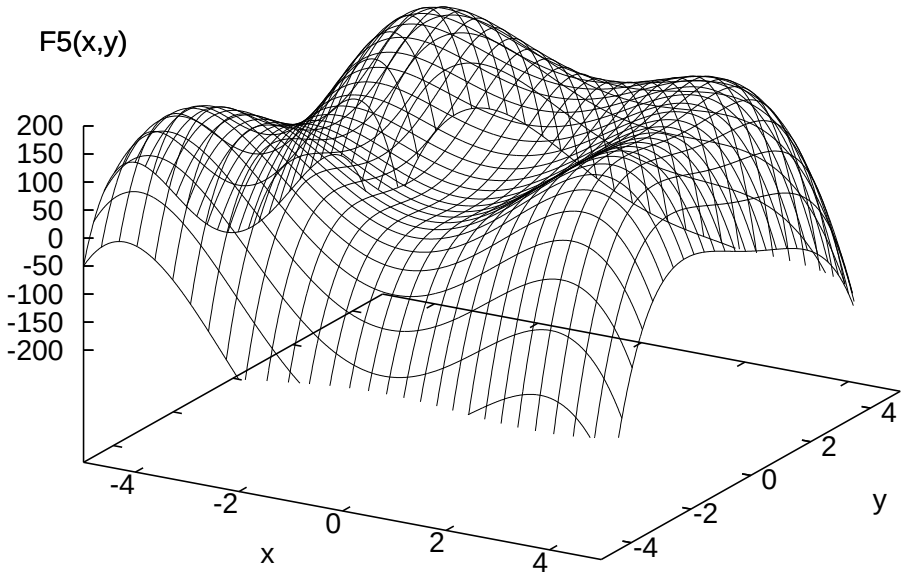


Figure 4.15: Function $F5$ (Himmelbrau).

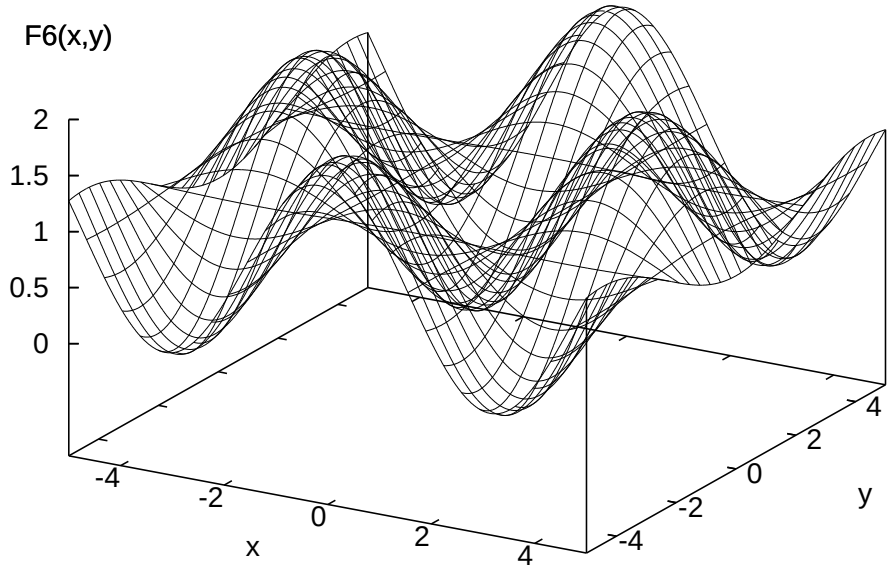


Figure 4.16: Function $F6$ (Griewank).

- Function $F5$, also known as the Himmelbrau function. It is defined by the equation:

$$F5(x,y) = 200 - (x^2 + y - 11)^2 - (x + y^2 - 7)^2. \tag{4.12}$$

  It has been studied in the range $(x,y) \in [-5,5] \times [-5,5]$, in which it has 4 maxima (see Figure 4.15).

- Function $F6$, also known as the Griewank function, defined as:

$$F6(x,y) = \frac{x^2 + y^2}{4000} - cos(x)cos(\frac{y}{\sqrt{2}}) + 1. \tag{4.13}$$

  The range is the same as the Himmelbrau function, $(x,y) \in [-5,5] \times [-5,5]$. Here the Griewank function has 5 minima (see Figure 4.16).

- Function $F7$, the Rastrigin function. It can be defined by:

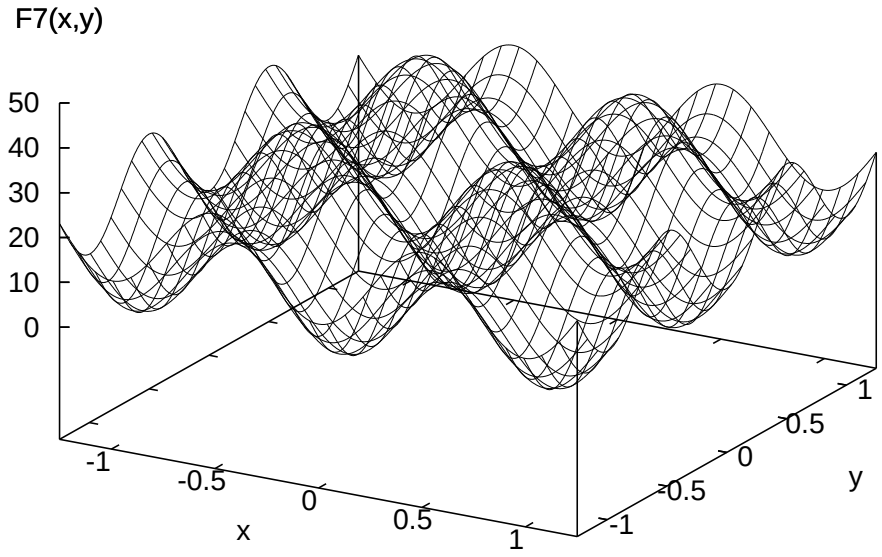


Figure 4.17: Function $F7$ (Rastrigin).

Table 4.2: Performance results. Figures indicate the number of experiments out of 100 in which the algorithm located all optima.

	% experiments locating all optima		
F	NichePSO	Parallel Vector-based PSO	kPSO
$F1$	100%	100%	100%
$F2$	100%	100%	100%
$F3$	100%	100%	100%
$F4$	100%	100%	100%
$F5$	100%	98%	100%
$F6$	100%	95%	98%
$F7$	100%	91%	99%

$$F7(x, y) = x^2 + y^2 - 10(cos(2\pi x) + cos(2\pi y)) + 20. \tag{4.14}$$

The Rastrigin function was studied in the range $(x, y) \in [-1.25, 1.25] \times [-1.25, 1.25]$, where it has 9 minima (see Figure 4.17).

The kPSO algorithm was applied using quite standard particle swarm coefficients, such as $c_1 = c_2 = 2.05$, resulting in $\chi \simeq 0.730$. Both the number of particles and the number of clusters were chosen to be proportional to the number of the expected optima of the function under consideration. In particular, let m be the number of optima for a given function, then kPSO was applied using N particles and k clusters such that:

$$N = 6m \qquad \text{and} \qquad k = \frac{3}{2}m. \tag{4.15}$$

Thus we used 30 particles for functions $F1$ to $F4$ and Griewank ($F6$), 24 particles for Himmelbrau ($F5$), and 54 for Rastrigin ($F7$). For all test functions the swarm was allowed to fly for 30 iterations, during which the clustering procedure was carried on every $c = 6$ steps.

4.5.2 Results

Results are summarized in Table 4.2, where the figures indicate the number of experiments in which the algorithm found all the optima on a total number

of 100 experiments for each function. The corresponding figures for the NichePSO [BEvdB02a] and the Parallel Vector-based PSO [SE05] are also reported.

The results show that kPSO is quite on par performance-wise with the other particle swarm niching algorithms. In particular, on the one-dimensional functions, all the algorithms could locate all the optima, thus achieving a figure of 100%. On the two-dimensional functions, instead, only the NichePSO approach could always achieve maximum performance. However, the kPSO outperformed the Parallel Vector-based PSO, although it achieved slightly worse results than NichePSO.

4.6 Un-niched particles

The experiments of the previous section proved that kPSO is quite competitive with respect to other PSO niching approach, but still needs some refinement. In particular, it could not find all the optima of some of the most complex functions of the benchmark set, unlike NichePSO.

One of the aspects of the algorithm which can be improved is the behavior of the un-niched particles. In the basic version of kPSO, in fact, the particles which are re-initialized after the application of the clustering procedure, perform a simple local search, behaving as in the *cognition only* model. It is possible that the poor capabilities of this model hinder the search of new solutions, leading to the less-than-optimal performance on some more complex functions.

In order to investigate this issue, we modified the kPSO algorithm so that the un-niched particles are organized in a more efficient social structure. We run experiments using the *lbest* neighborhood topology with varying values for k (the number of connections on the ring) and the von Neumann lattice neighborhood (see Figure 4.18). The particles which are re-initialized after each clustering application are then organized on the new topology. In particular for the von Neumann lattice, we arrange the N_u un-niched particles in rows of $\lfloor \sqrt{N_u} + \frac{1}{2} \rfloor$ particles, with the possibly uneven number of particles on the last row connected as in Figure 4.19.

The von Neumann topology showed the best results, allowing the algorithm to consistently locate all the solutions in every run. Thus we decided to use it in all subsequent experiments. In fact, we repeated the test performed in the previous section with the new topology, and obtained a success rate of 100% in locating all the optima even for the most complex functions.

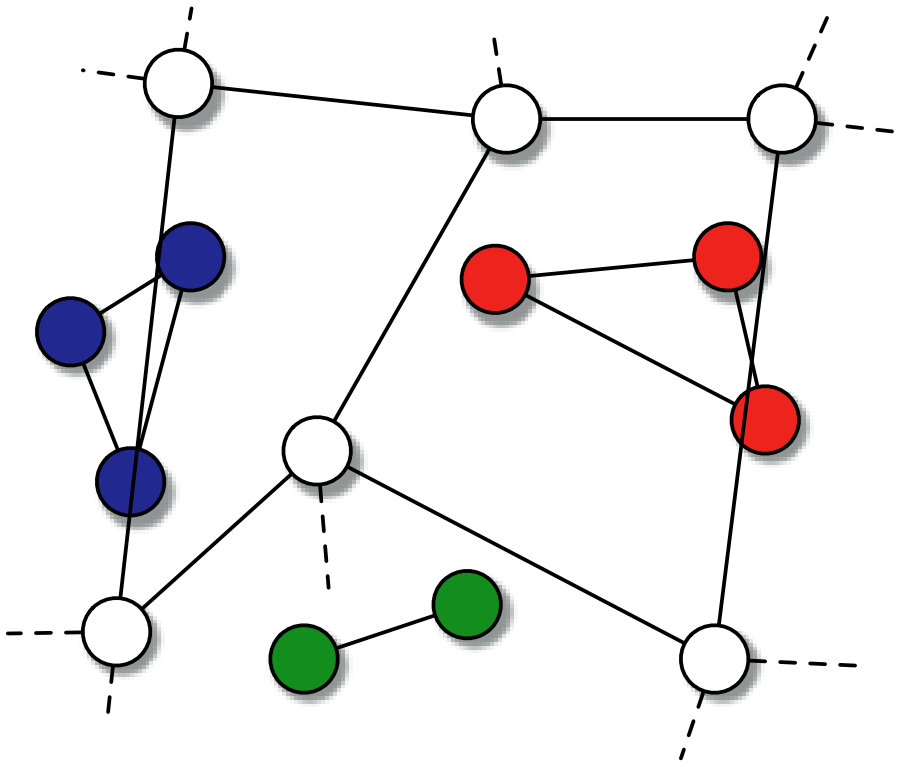


Figure 4.18: The particles not belonging to any niche (drawn in white) are organized in a von Neumann lattice topology.

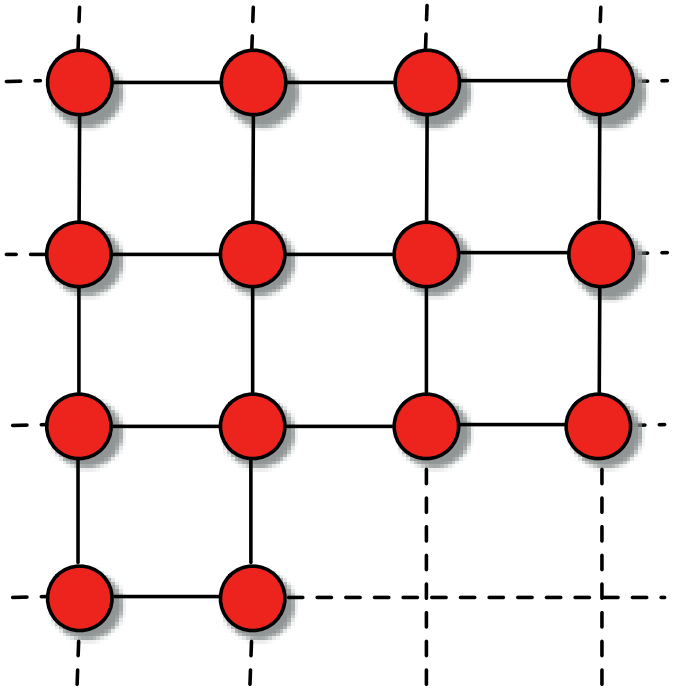


Figure 4.19: An example of a von Neumann lattice with an uneven number of particles. The connections of the particles on the last two rows are wrapped to the left or to the bottom as showed in figure.

Table 4.3: Number of evaluations required to locate all the optima of the respective functions. The average and standard deviation values over 100 runs are reported. The results in parentheses indicate the number of evaluations required to locate only the single *global* optimum of functions $F2$ and $F4$. The p-values for an algorithm refer to the t-test comparing its results with those of kPSO on the same function.

F	kPSO	NichePSO		SPSO	
			p-value		p-value
$F1$	$\mathbf{1382 \pm 243}$	1628 ± 288	< 0.001	2372 ± 109	< 0.001
$F2$	$\mathbf{1599 \pm 391}$ (674 ± 311)	2934 ± 475	< 0.001	$(\mathbf{352 \pm 202})$	< 0.001
$F3$	1501 ± 346	2404 ± 195	< 0.001	$\mathbf{1248 \pm 319}$	< 0.001
$F4$	$\mathbf{1530 \pm 395}$ (651 ± 318)	2820 ± 517	< 0.001	$(\mathbf{503 \pm 280})$	< 0.001
$F5$	$\mathbf{2032 \pm 340}$	$\mathbf{2151 \pm 200}$	0.003	3155 ± 402	< 0.001

4.6.1 Efficiency test

The new version of the kPSO algorithm with the un-niched particles organized on a von Neumann topology proved to be capable of matching NichePSO in its ability to locate all the optima of the functions in our benchmark set. At this point it is interesting to consider another important aspect of the performance of an optimization algorithm, that is its efficiency, which is usually measured in terms of the number of function evaluations required to locate all the optima with an accuracy of $\epsilon = 0.0001$.

In order to evaluate the efficiency, we repeated a selection of the experiments performed in Section 4.5. In particular, the main goal of the new experiments was to compare the performance of kPSO with those of NichePSO and of SPSO, respectively reported in [Bri02] and in [Li04]. An analysis of the statistical significance of the results was carried out by performing two t-tests between the results obtained with kPSO and respectively NichePSO and SPSO on the same function. The α-level was set to 0.001, thus accepting only very significant results.

The results are reported in Table 4.3, along with the p-values corresponding to the t-tests comparing the results of kPSO and the competing algorithm. In all the experiments, kPSO performed quite well, requiring a significantly smaller number of evaluations than NichePSO on all the functions except for $F5$, where the difference was not significant. The comparison with the other particle swarm niching algorithm

was more controversial, since SPSO performed better on functions $F1$ and $F3$, but worse on $F5$. Functions $F2$ and $F4$ are to be considered on their own, since they have 1 global optimum and 4 local ones and SPSO was set up to locate only the global optimum. In fact, the results reported in parentheses in Table 4.3, for both SPSO and kPSO, refer to the number of evaluations required to find just the global optimum.

4.7 Estimating the number of clusters

One of the major issues with our approach in kPSO is the fact that one needs to specify the number of clusters k, which is strongly related to the number of optima of the function under consideration. Unfortunately, often the number of optima of a function cannot be estimated *a priori*; rather, it would be desirable that it could be discovered by the optimization algorithm itself.

The need to know the number of clusters is inherited by kPSO from the specific clustering algorithm we chose, k-means, although it is a problem which is common to a wide class of clustering algorithms. Significant research has focused on the problem of estimating k, using very different approaches [XW05]. Here we will discuss an approach which leads to the estimation of k by the optimization of a criterion function in a probabilistic mixture-model framework.

4.7.1 Mixture Densities

In this framework, the objects to be clustered (the particles' position in our case) are assumed to be generated by several probabilistic distributions. In particular, to each cluster corresponds a different distribution. Thus, a general model for the whole set of objects will be given by a combination of different probability densities, the Mixture Densities [MP00], which can be defined as:

$$\rho(\mathbf{x}|\boldsymbol{\theta}) = \sum_{j=1}^{k} P(C_j)\rho(\mathbf{x}|C_j, \boldsymbol{\theta}_j). \tag{4.16}$$

In Equation (4.16), $P(C_j)$ is the prior probability of an object $\mathbf{x}$ to belong to the cluster C_j, with the condition that:

$$\sum_{j=1}^{k} P(C_j) = 1. \tag{4.17}$$

Instead, $\rho(\mathbf{x}|C_j, \boldsymbol{\theta}_j)$ is the conditional probability distribution associated with the same cluster (component density). Finally, $\boldsymbol{\theta} = (\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_k)$ is the vector of the parameters of the distributions for all clusters.

Assuming that the number of clusters k, the prior probabilities, and the form of each of the probability densities are known, it is possible to estimate the best parameters of a model by maximizing the probability of generating all the observations (Maximum Likelihood):

$$\rho(\{\mathbf{x}_1, \ldots, \mathbf{x}_N\}|\boldsymbol{\theta}) = \prod_{i=1}^{N} \rho(\mathbf{x_i}|\boldsymbol{\theta}), \tag{4.18}$$

or, in a logarithm form:

$$\ell(\boldsymbol{\theta}) = \sum_{i=1}^{N} \log \rho(\mathbf{x_i}|\boldsymbol{\theta}). \tag{4.19}$$

It has been proven [DH73], that the k-means algorithm implicitly performs a maximization of the likelihood under the assumption that the component densities are spherical Gaussian distributions.

Extending on these considerations, finding the number of clusters k is equivalent to fitting the model with the observed data and optimizing some criterion. This can be done by applying the k-means algorithm with k varying in a range of possible values and then choosing the best clustering.

However, the problem arises of choosing a valid criterion to select the best clustering. Here the maximum likelihood is not helping, since it invariably leads to the choice of the highest k. Thus, a large number of alternative criteria have been proposed, which combine concepts from information theory. The most typical examples include the Akaike's information criterion (AIC) [Aka74] [WC92] and the Bayesian Information Criterion (BIC) [Sch78] [PM00]. However, there is no criterion that is superior to others in general. The selection of different criteria is still dependent on the particular problem at hand.

4.7.2 The Bayesian Information Criterion

In our approach, we found that we had very similar results using either AIC or BIC, with a slightly better performance for the latter. Thus, we chose to integrate the Bayesian Information Criterion (BIC), also known as the Schwarz criterion, in order

to estimate the best choice of k. Given a clustering $\mathcal{C}_k$, formed by k clusters of a swarm with N particles, we can calculate its BIC value with:

$$BIC(\mathcal{C}_k) = \ell(\mathcal{C}_k) - \frac{p}{2} \cdot \log N, \tag{4.20}$$

where $\ell(\mathcal{C}_k)$ is the log-likelihood of the clustering and p is the number of parameters. The formula for the log-likelihood can be calculated considering that we are assuming components densities in the form of spherical Gaussians:

$$\rho(\mathbf{x}|\mathbf{m}_j, \sigma_j) = \frac{1}{\sqrt{2\pi}\sigma_j^d} e^{-\frac{1}{2\sigma_j^2}\|\mathbf{x}-\mathbf{m}_j\|^2}, \tag{4.21}$$

and the class probabilities are estimated with the ratios between the number of particles in a cluster and the total number of particles:

$$P(C_j) = \frac{N_j}{N}. \tag{4.22}$$

After a few mathematical transformations, the log-likelihood of the clustering, $\ell(\mathcal{C}_k)$, can be written as:

$$\ell(\mathcal{C}_k) = \sum_{i=1}^{N} \log \rho(\mathbf{x_i}|\mathcal{C}_k) = \sum_{j=1}^{k} \ell(C_j) - N \cdot \log N, \tag{4.23}$$

where the term $\ell(C_j)$ is the log-likelihood for each cluster C_j:

$$\ell(C_j) = -\frac{N_j}{2} \cdot \log 2\pi - \frac{N_j \cdot d}{2} \cdot \log \sigma_j^2 - \frac{N_j - 1}{2} + N_j \cdot \log N_j. \tag{4.24}$$

The number of parameters p is given by the sum of $k-1$ class probabilities (given the condition in Equation (4.17)), $d \cdot k$ centroid coordinates, and the k variance estimates σ_j. Thus we have:

$$p = (k-1) + d \cdot k + k. \tag{4.25}$$

In the improved version of the kPSO algorithm, at each clustering application, k-means is thus repeated with varying values for k, then the clustering with the highest BIC is chosen. In this way there is no need to set a value for k at the beginning of the run. Rather, it can vary across the execution of the algorithm, as the particles in the swarm slowly discover new promising areas and organize in new niches.

4.8 Experiments with the improved kPSO

In the previous series of tests, we showed how the basic version of the kPSO algorithm, without the automatic selection of the number of clusters, could outperform existing algorithm such as NichePSO and Parallel Vector-based PSO. In general, however, its performance was not as good as that of SPSO. In this section we will set up a new experiment in which we will compare the improved kPSO, which can adaptively identify the number of clusters using BIC, to the SPSO algorithm and also to the ANPSO algorithm.

The relationship between ANPSO and SPSO is in a way similar to that between the improved kPSO and its basic version. In fact, while SPSO does not require in input the number of clusters, it does need to know *a priori* some information about the function to optimize, specifically, the niche radius σ. Thus, the same author of SPSO, Li, contributed to the introduction of ANPSO, specifically to overcome this limitation, as the latter algorithm can adaptively determine the niche radius.

4.8.1 Benchmark functions

The study was conducted on a new benchmark set of multimodal functions, chosen to be consistent with the one used in [BL06] to compare the performance of SPSO and ANPSO. In the following, we describe the multimodal functions on which the experiments have been carried out.

Branin RCOS

The first function in the study was the Branin RCOS function, a two-dimensional function defined as:

$$M1(x,y) = \left(y - \frac{5.1x^2}{4\pi^2} + \frac{5x}{\pi} - 6\right)^2 + 10\left(1 - \frac{1}{8\pi}\right)\cos(x) + 10. \tag{4.26}$$

It was studied in the range $(x,y) \in [-5,10] \times [0,15]$, where it has 3 minima, as plotted in Figure 4.20.

Six-Hump Camel Back

The second function was the Six-Hump Camel Back, plotted in Figure 4.21, and defined as:

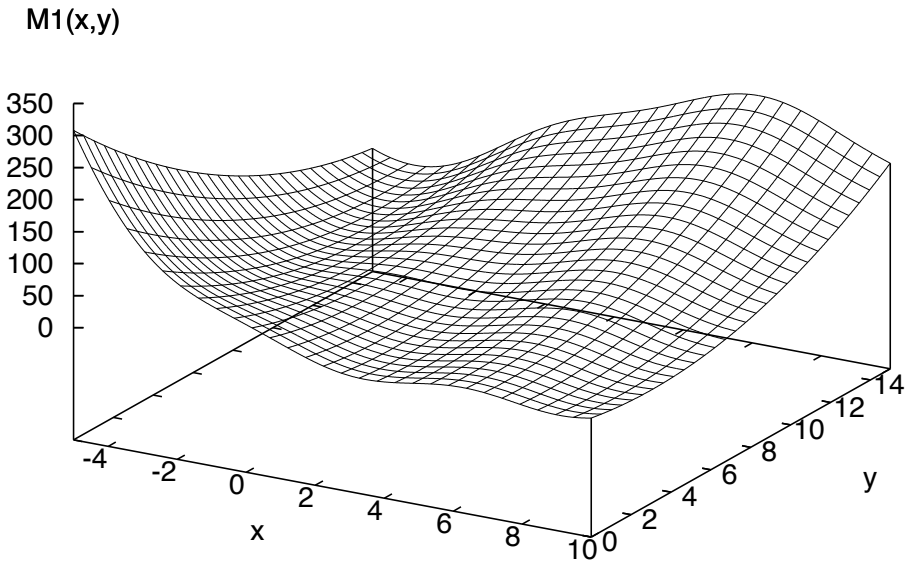


Figure 4.20: Function $M1$ (Branin RCOS).

$$M2(x,y) = -4\left[\left(4 - 2.1x^2 + \frac{x^4}{3}\right)x^2 + xy + \left(-4 + 4y^2\right)y^2\right]. \tag{4.27}$$

It was studied in the range $(x,y) \in [-1.9, 1.9] \times [-1.1, 1.1]$, where it has 2 global maxima, but several deceptive local ones.

Deb's 1^{st} function

The third function, the Deb's 1^{st} function, was the same simple mono-dimensional function $F1$ used in the experiments in Section 4.5. It has with 5 equally spaced global maxima in the considered range $x \in [0,1]$ (see Figure 4.11). The Deb's 1^{st} function was defined in Equation (4.8). We report it here for clarity:

$$M3(x) = F1(x) = \sin^6(5\pi x). \tag{4.28}$$

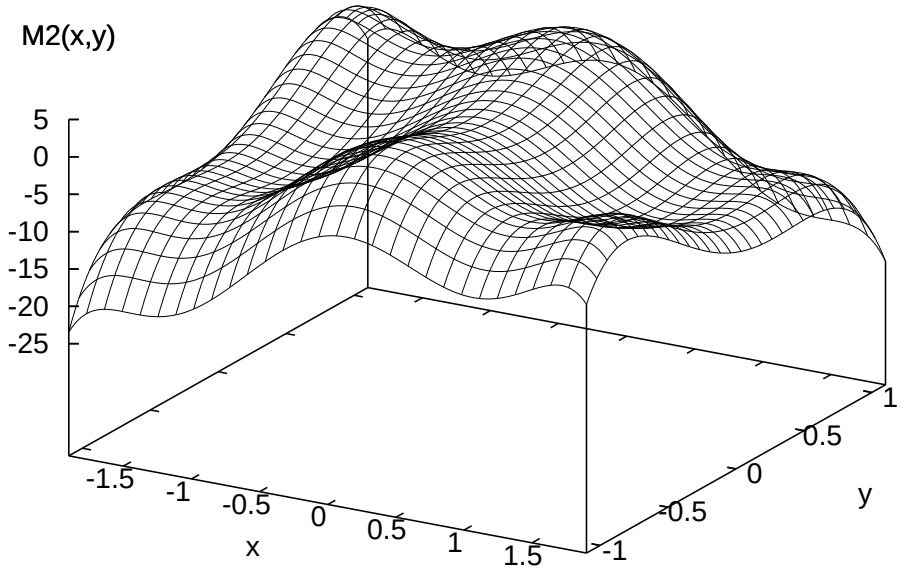


Figure 4.21: Function $M2$ (Six-Hump Camel Back).

Himmelblau

The fourth function was the Himmelblau function, again the same used as $F5$ in Section 4.5 (Figure 4.15). It is a two-dimensional function, defined as (see also Equation (4.12)):

$$M4(x, y) = F5(x, y) = 200 - (x^2 + y - 11)^2 - (x + y^2 - 7)^2. \tag{4.29}$$

In these experiments, it was studied in the slightly extended range $(x, y) \in [-6, 6] \times [-6, 6]$. However, this does not actually change anything, since in this range it still has the same 4 maxima.

Shubert 2D

Finally, the fifth and last function of the study was the Shubert 2D function, with 18 global optima grouped in 9 clusters and surrounded by a very high number of local optima (Figure 4.22). The Shubert 2D function is defined as:

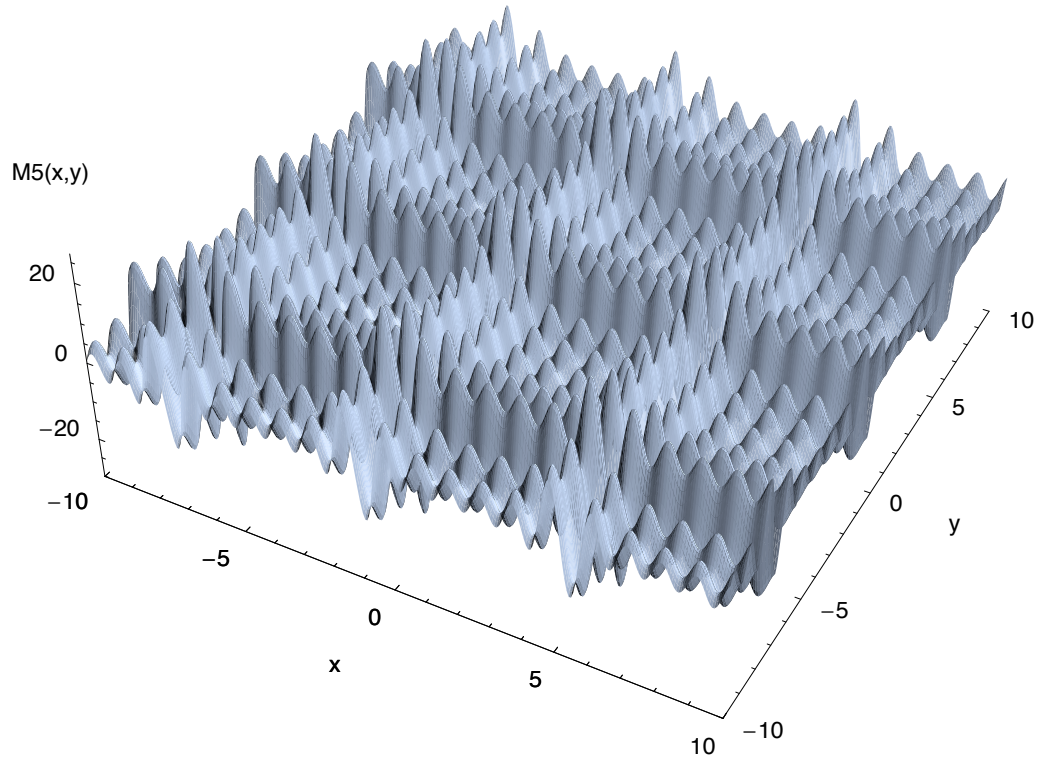


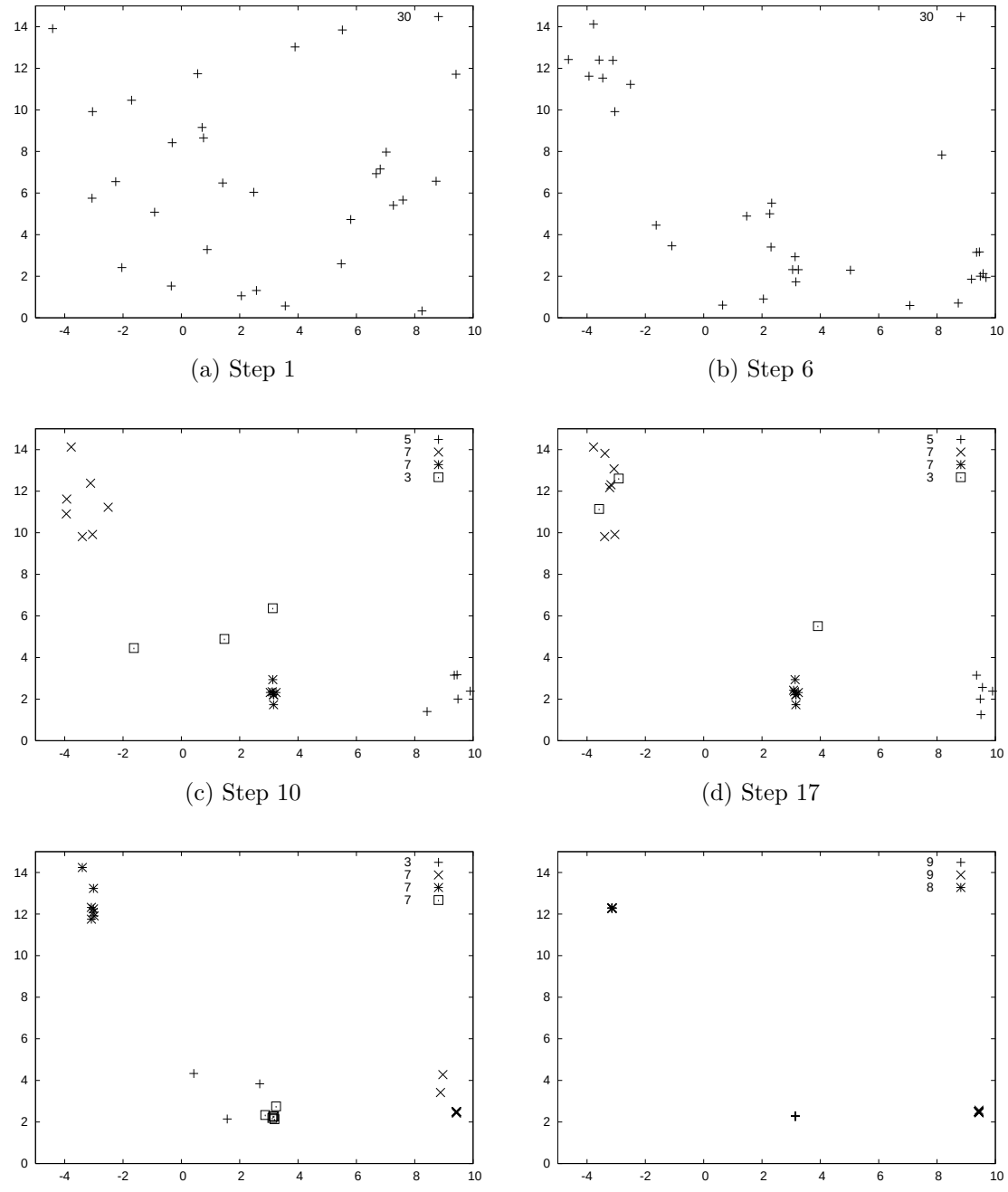
Figure 4.22: Function $M5$ (Shubert 2D).

$$M5(x,y) = \sum_{i=1}^{5} i\cos[(i+1)x+i] \sum_{i=1}^{5} i\cos[(i+1)y+i], \tag{4.30}$$

and it was studied in the range $(x,y) \in [-10,10] \times [-10,10]$.

4.8.2 Running kPSO

In order to show how the improved version of our algorithm can effectively identify niches surrounding the optima of a function, we report in Figure 4.23 several significant steps of kPSO running on the Branin RCOS function $M1$. In the snapshots we plotted, it is shown how, at the beginning of the run, the particles of the swarm are randomly distributed on the search space. Then, during the first steps of the particle swarm simulation, they naturally start to split in different



(a) Step 1

(b) Step 6

(c) Step 10

(d) Step 17

(e) Step 41

(f) Step 65

Figure 4.23: Significant steps of a kPSO run on function $M1$. In the first step particles are randomly distributed on the search space. At step 6, the swarm is naturally splitting in different groups of particles around the global optima of the function. At step 10, the first clustering is performed, which identifies 4 niches. Step 17 shows how particles belonging to different niches can get mixed up when their niches are actually related to the same global optimum. Finally, at step 65, the algorithm stabilizes on 3 niches corresponding to the 3 global optima.

groups, roughly adapting to the fitness landscape.

When the first clustering application occurs (see Figure 4.23c), it identifies 4 niches, of which 3 actually correspond to the global optima, whilst the other is a spurious one. In the particular case that is shown here, the algorithm will eventually converge to exactly 3 clusters, as it is shown in Figure 4.23f, relative to the 62^{nd} simulation step. However, such convergence is not the final goal of kPSO: as long as a sufficient number of clusters has formed to cover all the optima, the presence of spurious clusters is really of no harm to the optimization algorithms. Besides, in early phases of the optimization, the presence of additional clusters can be helpful in locating new interesting regions of the search space.

4.8.3 Results

In Table 4.4 we report the results obtained on the five benchmark functions with kPSO, SPSO, and ANPSO. The experiments were carried out so that for each function the optimization algorithm was executed with two different population sizes (N). Each execution was repeated 50 times, setting the threshold for the maximum number of iteration to 2000 simulation steps.

In all the experiments, the goal of the optimization algorithm was to locate all the global optima with an accuracy of $\epsilon = 0.00001$ and to maintain them for at least 10 simulation steps. Since all the algorithms could actually fulfill the goal within the maximum number of 2000 iterations, we only report in Table 4.4 their performance in term of the number of function evaluations they required.

The first four functions in the benchmark, $M1, \ldots, M4$, proved to be quite easy to optimize. A population size of just 30 particles was more than sufficient to identify all the global optima with a quite low number of function evaluations. In Table 4.4, we report also the results with a population of 60 particles in order to compare them to those in [BL06]. kPSO proved to have a clear advantage on all of these functions with respect to SPSO and ANPSO. In particular, we performed a t-test between the results obtained with kPSO and the other two algorithms and it showed that the performance advantage of kPSO is significant within an α-level of 0.001.

In the experiments with functions $M1$ to $M4$ we used a value of $c = 10$ for the number of steps between two clustering applications. With higher values the performances of the algorithm did not vary significantly, meaning that it was a sufficient number of steps for the particles to adjust to the changed social structure after a clustering application.

Table 4.4: Number of evaluations required to find all global optima of the benchmark functions with different population sizes (N). The average and standard deviation values over 50 runs are reported. The p-values for an algorithm refer to the t-test comparing its results with those of kPSO on the same function.

F	N	kPSO	SPSO	p-value	ANPSO	p-value
$M1$	30	$\mathbf{2084 \pm 440}$	3169 ± 692	< 0.001	5220 ± 3323	< 0.001
	60	$\mathbf{3688 \pm 717}$	6226 ± 1707	< 0.001	6927 ± 2034	< 0.001
$M2$	30	$\mathbf{1124 \pm 216}$	2872 ± 827	< 0.001	2798 ± 857	< 0.001
	60	$\mathbf{2127 \pm 341}$	5820 ± 1469	< 0.001	4569 ± 1316	< 0.001
$M3$	30	$\mathbf{1207 \pm 688}$	2007 ± 703	< 0.001	6124 ± 2465	< 0.001
	60	$\mathbf{1654 \pm 705}$	4848 ± 2092	< 0.001	8665 ± 2974	< 0.001
$M4$	30	$\mathbf{2259 \pm 539}$	4096 ± 731	< 0.001	16308 ± 13157	< 0.001
	60	$\mathbf{3713 \pm 570}$	7590 ± 2018	< 0.001	17168 ± 12006	< 0.001
$M5$	300	$\mathbf{81194 \pm 45646}$	166050 ± 42214	< 0.001	$\mathbf{82248 \pm 10605}$	0.874
	500	$\mathbf{117503 \pm 77451}$	219420 ± 80179	< 0.001	$\mathbf{114580 \pm 18392}$	0.796

A special analysis was conducted regarding the optimization of function $M5$, which was the most complex function of the set, in particular because it presented many local optima surrounding the global ones. As showed in Figure 4.24, even if kPSO was able to find all the optima in all the runs, the number of evaluations required changed greatly depending on the value of c. We found that a good value was $c = 50$, so we used it for the other experiments (Table 4.4 and Figure 4.25).

The higher degree of complexity of function $M5$ also led to the use of a much larger population. In the experiments reported in Table 4.4, we employed the two population sizes of 300 and 500 in order to compare the results to those of ANPSO and SPSO as reported in [BL06]. The performance of kPSO appears to be significantly better than those of SPSO and comparable to those of ANPSO, although they exhibit a larger variance over the 50 runs. However, kPSO successfully located all the optima in $M5$ also with smaller population sizes, as plotted in Figure 4.25, requiring rather fewer function evaluations than the other algorithms. In particular, with a population of only 200 particles, kPSO was able to locate all the global optima performing just 59165 ± 32646 function evaluations.

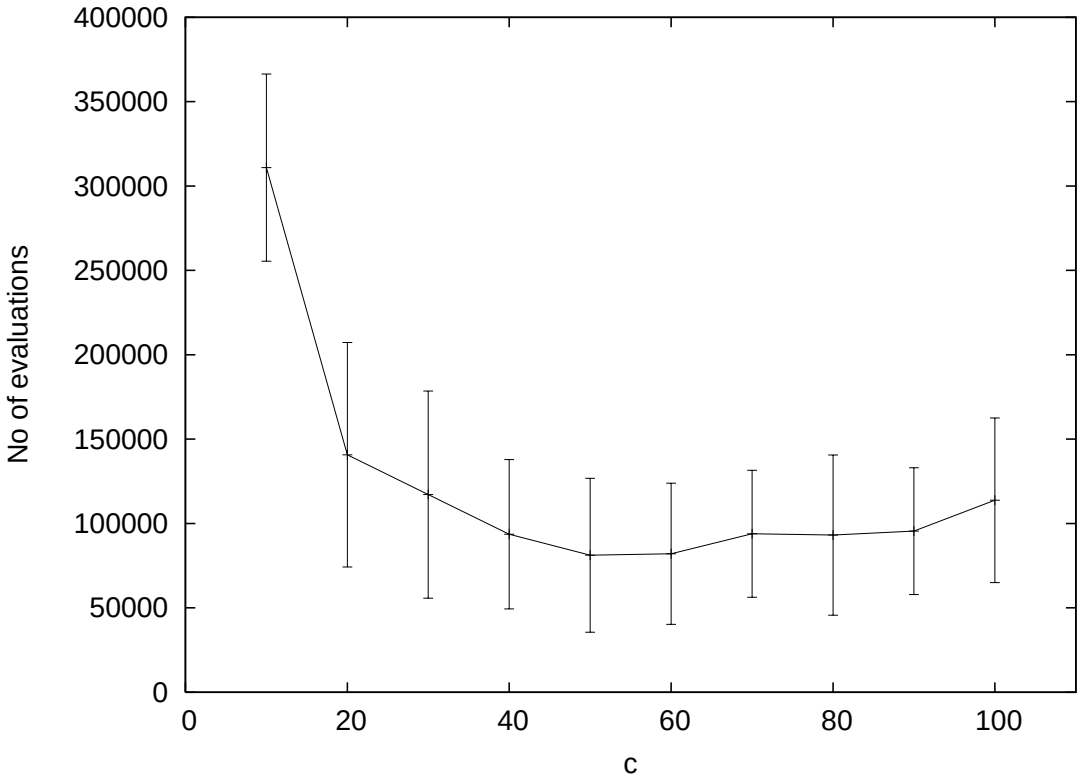


Figure 4.24: Performance of kPSO on function $M5$ using different values for c, the number of steps between clusterings, and a fixed population size of $N = 300$. Error bars represent the standard deviation.

4.9 Discussion

The last set of experiments was intended to test the performance of the improved version of kPSO and to compare it with two of the best existing niching PSO algorithms, SPSO and ANPSO. The results we obtained were quite good: the kPSO algorithm, with the new mechanism to adaptively determine the number of clusters, outperformed by a good margin the other algorithms on most of the test functions.

The situation was rather more elaborated regarding the most complex function of the group, the Shubert 2D function. In this case, kPSO still showed very good results on average, but also a very large variance. Moreover it needed an adjustment of the newly introduced parameter, the number of simulation steps between two clustering applications. In fact, while for the simple functions this number was kept constant at a quite low value, the application to the Shubert function required a much higher value, in a way that is consistent with what we found out in earlier study on the

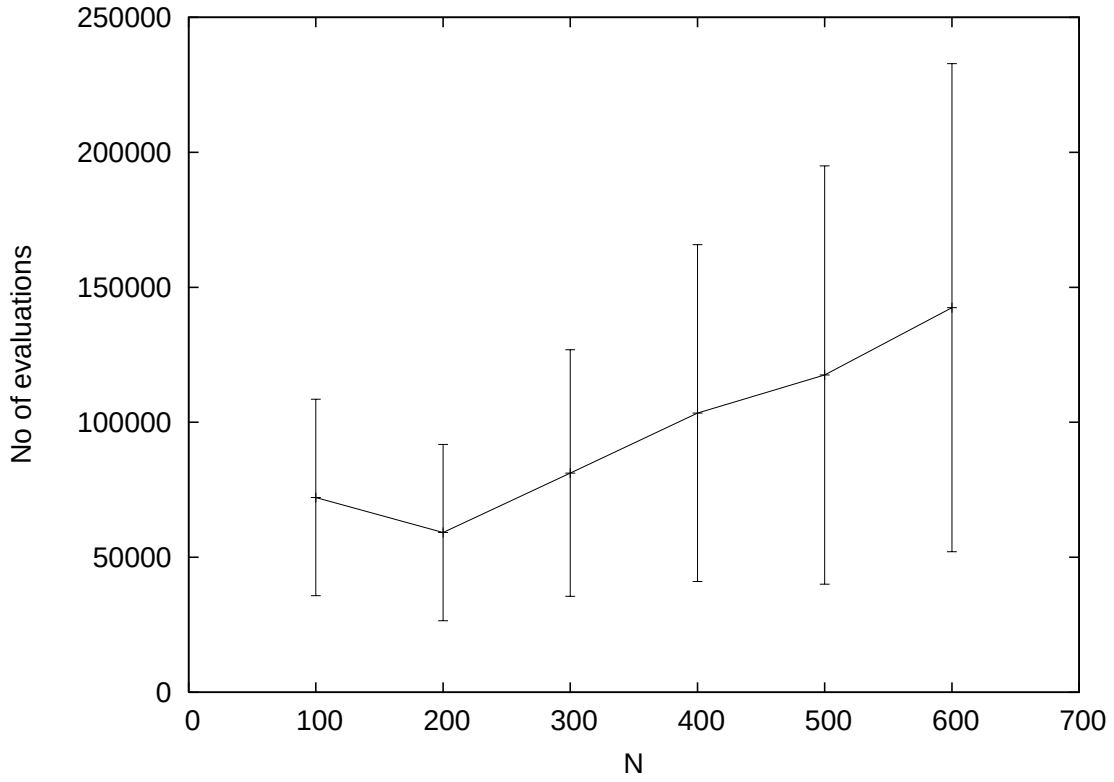


Figure 4.25: Performance of kPSO on function $M5$ using different population sizes N, and a fixed value of $c = 50$ for the number of steps between two clustering applications. Error bars represent the standard deviation.

influence of c (see Section 4.4.1).

Another important aspect to consider in the evaluation of the different approaches to niching is the computational cost of the niching procedure itself. In general, when considering the efficiency of an optimization algorithm, the most significant measure is the one we used in our last experiments, that is the number of function evaluations the algorithm requires to reach the predefined goal. In fact, calculating the function value is usually the single operation with the highest computational cost in a real-world problem.

However, in certain cases, the computational overhead added by other operations required by the optimization algorithm should be taken into account. In the comparison we are discussing, all the algorithms implement a version of the particle swarm approach, thus they share roughly the same (quite low) computational costs related to the simulation of particles' dynamics. In this regard, the only aspect in which they differ is the specific niching procedure they use.

Table 4.5: Computational overhead of the niching procedure employed in different algorithms, averaged per single iteration. N is the swarm size, k the number of niches, r_k the number of runs of k-means, and c the number of steps before clustering.

Algorithm	Cost
SPSO	$O(N \cdot k)$
ANPSO	$O(N^2)$
kPSO	$O(N^2 \cdot r_k/c) \simeq O(N^2)$

In particular, SPSO uses the procedure to identify the species seeds which is reported in Algorithm 3.5, ANPSO the niche formation procedure in Algorithm 3.7, and kPSO the k-means based procedure in Algorithm 4.2, enhanced with the automatic selection of the best number of clusters reported in Section 4.7.2. The computational cost of this procedures can be evaluated in terms of the number of distance calculations. For the SPSO algorithm, this leads to a computational cost at each iteration that is $O(N \cdot k)$, where N is the swarm size and k the number of seeds or niches. The ANPSO algorithm, instead, implements a more complex procedure, which has a cost of $O(N^2)$.

Our approach basically inherits the computational complexity of the k-means algorithm, that is $O(N \cdot k \cdot r_k)$. But considering that the improved version actually repeats the k-means application in order to determine the best number of clusters, the final computational cost is in the order of $O(N^2 \cdot r_k)$. It should be noted, however, that kPSO, unlike the other algorithms, only executes the niching procedure every c simulation steps, thus leveraging its computational overhead by the same factor. Considering the order of magnitude of the parameters r_k and c we used in all the experiments, the cost per iteration is $O(N^2 \cdot r_k/c) \simeq O(N^2)$, therefore in the same order as ANPSO. A summary of the computational overhead of the various niching procedure, averaged per single iteration, is given in Table 4.5.

Whilst in this analysis it emerges that the SPSO algorithm is the one which adds the lowest computational overhead to identify niches, compared to ANPSO and kPSO, whether this is really important or not depends mainly on two factors:

1. In many real-world optimization problems, the most relevant component of the computational cost is the number of function evaluations. Thus, an algorithm with an highest overhead in term of distance calculations, but which requires

significantly less evaluations is surely to be preferred.

2. Fine-tuning parameters adds another computational cost. This is just one of the main issues Bird and Li tried to assess with the introduction of ANPSO over SPSO. The latter, in fact, requires the specification of a fixed niche radius, thus it realistically needs to be executed multiple times to find the best value, while ANPSO can adaptively determine it during a single run. In this regard, kPSO is more similar to ANPSO, as it can adaptively determine k, the most problem dependent parameter. However, it still needs a value for c to be chosen, which can significantly influence its efficiency, although does not really hinder its ability to locate all the optima.

Chapter 5

Conclusions

The research we presented in this thesis originated from two fundamental topics, which we treated in the first two chapters.

The first topic is the particle swarm algorithm, an optimization technique which was developed starting from artificial life simulations and is now an effective alternative to more established methods, like those in the evolutionary computation field.

The second topic is related to the optimization of multimodal functions, and thus to the development of optimization algorithms which can look for multiple solutions. In particular, our interest is focused on niching techniques, inspired from the tendency of natural species to occupy different niches of the environment. Niching techniques have been introduced in the context of evolutionary computation precisely with the intent to enable evolutionary algorithms to deal with multimodal functions.

The same path can be taken to modify the particle swarm algorithm in order to optimize multimodal functions. In this thesis we reviewed existing approaches to niching for PSO and then presented our new approach, which is based on clustering. The particles in the swarm, in fact, naturally tend to group around optimal regions of the function landscape. By applying a clustering algorithm, we can identify promising niches and use the swarm to locate multiple solutions.

We implemented the clustering approach, using the popular k-means clustering algorithm, in the kPSO algorithm. In this thesis, we described the development of the new algorithm from its first implementation to a more effective version, pointing out the issues we wanted to address at each step.

The new algorithm was put to test on some functions which have been built

in order to investigate kPSO ability to locate multiple optima. Successive tests, instead, were performed on commonly used benchmark functions, which allowed for a comparison with other niching algorithms for the particle swarm. The results of the tests were also used to analyze the aspects of the algorithm which could be improved. In particular, the first modification we made to kPSO involves the way it manages un-niched particles. In fact, in the first implementation as well, we employed a mechanism which removes particles from overcrowded clusters, randomly re-initializes them, and leaves them outside of any niche. The role of these particles is to search for interesting regions that are not covered by existing niches. In our first trials, however, they were left to wander on the search space without any interactions with other particles, following the so-called *cognition only* model. Successive experiments showed that a much more efficient path is to constrain them on a particular kind of neighborhood structure, the von Neumann topology, which allowed the kPSO algorithm to consistently locate all the optima of the functions in the benchmark set.

At this point of the development, kPSO proved to outperform other niching algorithms for PSO, such as NichePSO and PVPSO. In our experiments, in fact, it showed that it could locate all the optima of the functions under consideration more consistently in multiple runs and / or with fewer function evaluations. In the same tests, we compared kPSO also with SPSO, one of the most successful niching approach for the particle swarm. In this case, the results were more ambiguous, since the two algorithms performed differently in different tests. Nonetheless, kPSO showed to be a competitive alternative.

In the comparison between different optimization algorithms, an important aspect to consider is the influence of problem dependent parameters. An ideal algorithm, in fact, would be one that can be applied to a wide range of problems without having to fine-tune its parameters for each of them. Conversely, when the algorithm needs to be adapted for its application to new functions, the additional computational cost must be taken into consideration.

The kPSO algorithm uses essentially two critical parameters which are highly problem dependent. The first one is c, the number of simulation steps between two clustering applications. Our experiments showed that, although different values of c influence the performance of the algorithm in term of number of function evaluations, they could be chosen in a wide range without hampering the ability to locate all the optima. Moreover, while setting c to rather high values indeed worsened the performance of kPSO, it also lowered the computational overhead due

to the clustering. Thus, tuning c is essentially a matter of selecting the best balance between the costs of function evaluations and of clustering applications.

The second parameter which depends on the function under consideration is k, the number of clusters, that must be specified *a priori* to employ the k-means algorithm. Setting k can be much more troubling, since it strongly depends on the number of optima of the multimodal function. Especially in real-world applications, it is rarely possible to effectively determine a good value for k before the optimization. It should be noted, however, that most niching algorithms suffer from a rather similar problem. In many cases, such as for SPSO, the niching procedure needs setting the niche radius σ, which requires some knowledge about the *width* of the niches, rather than their number. Nonetheless, tuning σ is a rather equally problematic issue.

The final version of our kPSO algorithm enhances the clustering process by incorporating a mechanism to adaptively determine the number of clusters k. Essentially, the algorithm repeats the application of the k-means clustering varying the value of k and then selects the *most natural* among the obtained clusterings, by choosing the one with the highest value for the Bayesian Information Criterion. The price to pay is a higher computational overhead, but the improved kPSO gains in immediate applicability to different problems and also in better performance, due to the ability to adapt the number of clusters during the run.

In conclusion, the clustering approach to niching we propose in this thesis proved to be rather effective. Its implementation in the final version of the kPSO algorithm was competitive with other niching algorithms. In particular, it markedly outperformed one of the best existing algorithms, SPSO, in term of the number of function evaluations needed to discover all the optima of the test functions. Even though the computational cost of the clustering procedure in kPSO is higher than that of SPSO, we thought that it is balanced by the better ability to adapt without manual tuning to any function landscape, other than by the better performance. In this respect, the advantages of kPSO are rather similar to those of ANPSO, another interesting algorithm which adaptively determines the main niching parameters. The two algorithms, in fact, introduce a comparable computational overhead with the procedure to identify niches. kPSO, however, showed a better or comparable performance in all the test we conducted.

Research on the clustering approach will continue in several directions. To begin with, the kPSO algorithm will be put to test with higher-dimensionality benchmark functions together with real-world problems, in order to better assess its capabilities.

Another interesting research line involves the development of a more flexible version of kPSO, which will avoid the need to set c, the number of steps between clusterings. This could be accomplished for example by developing an algorithm which could estimate when a new clustering application is needed, rather than performing it at fixed intervals. Further research will also be devoted to investigate the employment of different clustering algorithms rather than the k-means.

List of Tables

List of Figures

List of Algorithms

Bibliography

[Aka74] H. Akaike, *A new look at the statistical model identification*, IEEE Transactions on Automatic Control **19** (1974), no. 6, 716–723.

[Ang98a] P. J. Angeline, *Evolutionary optimization versus particle swarm optimization: Philosophy and performance differences*, EP '98: Proceedings of the 7th International Conference on Evolutionary Programming VII, Springer-Verlag, 1998, pp. 601–610.

[Ang98b] ———, *Using selection to improve particle swarm optimization*, IEEE International Conference on Evolutionary Computation (Anchorage, Alaska), May 1998.

[Bak87] James E. Baker, *Reducing bias and inefficiency in the selection algorithm*, Proceedings of the Second International Conference on Genetic Algorithms on Genetic algorithms and their application (Mahwah, NJ, USA), Lawrence Erlbaum Associates, Inc., 1987, pp. 14–21.

[Bak96] P. Bak, *How nature works*, 1 ed., Copernicus, Springer-Verlag, New York, 1996.

[BBM93] D. Beasley, D. R. Bull, and R. R. Martin, *A sequential niche technique for multimodal function optimization*, Evolutionary Computation **1** (1993), no. 2, 101–125.

[BDT99] E. W. Bonabeau, M. Dorigo, and G. Theraulaz, *Swarm intelligence: from natural to artificial systems*, Oxford University Press, 1999.

[BEvdB02a] R. Brits, A.P. Engelbrecht, and F. van den Bergh, *A niching particle swarm optimizer*, Proceedings of the 4th Asia-Pacific Conference on Simulated Evolution and Learning (SEAL'02) (Singapore) (L. Wang,

K. C. Tan, T. Furuhashi, J. H. Kim, and X. Yao, eds.), vol. 2, November 2002, pp. 692–696.

[BEvdB02b] R. Brits, A.P. Engelbrecht, and F. van den Bergh, *Solving systems of unconstrained equations using particle swarm optimization*, Proceedings of the IEEE 2002 Conference on Systems, Man, and Cybernetics (Tunisia), 2002.

[BEvdB03] R. Brits, A.P. Engelbrecht, and F. van den Bergh, *Scalability of niche pso*, Proceedings of the IEEE Swarm Intelligence Symposium (SIS 2003) (Indianapolis, Indiana, USA), 2003, pp. 228–234.

[Bis95] C. M. Bishop, *Neural networks for pattern recognition*, Oxford University Press, Inc., 1995.

[BL06] Stefan Bird and Xiaodong Li, *Adaptively choosing niching parameters in a pso*, GECCO '06: Proceedings of the 8th annual conference on Genetic and evolutionary computation (New York, NY, USA), ACM Press, 2006, pp. 3–10.

[BPV02] T. Beielstein, K. Parsopoulos, and M. Vrahatis, *Tuning pso parameters through sensitivity analysis*, Tech. report, Collaborative Research Center 531 Computational Intelligence CI, University of Dortmund, January 2002.

[Bri02] R. Brits, *Niching strategies for particle swarm optimization*, Master's thesis, University of Pretoria, 2002.

[BTW87] Per Bak, Chao Tang, and Kurt Wiesenfeld, *Self-organized criticality: An explanation of the 1/f noise*, Phys. Rev. Lett. **59** (1987), no. 4, 381–384.

[Bur70] A. W. Burks (ed.), *Essays on cellular automata*, University of Illinois Press, Urbana, Illinois, 1970.

[Cav70] D. J. Cavicchio, *Adaptive search using simulated evolution*, Tech. Report Report 03296-4-T, Computer and Communication Sciences Department, University of Michigan, Ann Arbor, Michigan, 1970.

[CD01] A. Carlisle and G. Dozier, *An off-the-shelf PSO*, Proceedings of the Workshop on Particle Swarm Optimization (Indianapolis), 2001.

[CK02] M. Clerc and J. Kennedy, *The particle swarm - explosion, stability, and convergence in a multidimensional complex space*, IEEE Transactions on Evolutionary Computation **6** (2002), no. 1, 58–73.

[CRHW04] B. C. H. Chang, A. Ratnaweera, S. K. Halgamuge, and H. C. Watson, *Particle swarm optimisation for protein motif discovery*, Genetic Programming and Evolvable Machines **5** (2004), no. 2, 203–214.

[Dav91] Y. Davidor, *A naturally occurring niche and species phenomenon: The model and first results*, Proceedings of the 4th International Conference on Genetic Algorithms (San Mateo, CA) (R. Belew and L. Booker, eds.), Morgan Kaufmann, 1991, pp. 257–263.

[Daw76] R. Dawkins, *The selfish gene*, Oxford University Press, New York, 1976.

[DC99] Marco Dorigo and Gianni Di Caro, *The ant colony optimization meta-heuristic*, 11–32.

[dCT02] L. N. de Castro and J. Timmis, *Artificial immune systems: a new computational intelligence approach*, Springer, 2002.

[DG89] Kalyanmoy Deb and David E. Goldberg, *An investigation of niche and species formation in genetic function optimization*, Proceedings of the third international conference on Genetic algorithms (San Francisco, CA, USA), Morgan Kaufmann Publishers Inc., 1989, pp. 42–50.

[DG97] M. Dorigo and L.M. Gambardella, *Ant colony system: a cooperative learning approach to the travelingsalesman problem*, IEEE Transactions on Evolutionary Computation **1** (1997), no. 1, 53–66.

[DH73] R. O. Duda and P. E. Hart, *Pattern Classification and Scene Analysis*, Wiley, 1973.

[DJ75] Kenneth Alan De Jong, *An analysis of the behavior of a class of genetic adaptive systems.*, Ph.D. thesis, University of Michigan, Ann Arbor, MI, 1975.

[DY95] P. J. Darwen and X. Yao, *A Dilemma for Fitness Sharing with a Scaling Function*, Proceedings of the Second IEEE International Conference on Evolutionary Computation (Piscataway, New Jersey), IEEE Press, 1995.

[DY97] ———, *Speciation as automatic categorical modularization*, IEEE Transactions on Evolutionary Computation **1** (1997), 101–108.

[EGEHSK02] A. El-Gallad, M. El-Hawary, A. Sallam, and A. Kalas, *Enhancing the particle swarm optimizer via proper parameters selection*, IEEE Canadian Conference on Electrical and Computer Engineering (CCECE 2002), vol. 2, 2002, pp. 792–797.

[EH99] R.C. Eberhart and X. Hu, *Human tremor analysis using particle swarm optimization*, Proceedings of the Congress on Evolutionary Computation (CEC 99), 1999, pp. 1927–1930.

[EK95] R. C. Eberhart and J. Kennedy, *A new optimizer using particle swarm theory*, Proceedings of the Sixth International Symposium on Micro Machine and Human Science (MHS '95) (Nagoya, Japan), Oct 1995, pp. 39–43.

[EMP05] A.P. Engelbrecht, B.S. Masiye, and G. Pampara, *Niching ability of basic particle swarm optimization algorithms*, Proceedings of the IEEE 2005 Swarm Intelligence Symposium (SIS 2005) (Pasadena, California, U.S.A.), 2005, pp. 397–400.

[ES98a] R. C. Eberhart and Y. Shi, *Comparison between genetic algorithms and particle swarm optimization*, EP '98: Proceedings of the 7th International Conference on Evolutionary Programming VII, Springer-Verlag, 1998, pp. 611–616.

[ES98b] R. C. Eberhart and Y. Shi, *Evolving artificial neural networks*, Proceedings of the International Conference on Neural Networks and Brain (Beijing, P.R.C.), 1998.

[ESD96] R. C. Eberhart, P. K. Simpson, and R. W. Dobbins, *Computational intelligence pc tools*, Academic Press Professional, Boston, MA, 1996.

[FE03] N. Franken and A. P. Engelbrecht, *Evolving intelligent game-playing agents*, SAICSIT '03: Proceedings of the 2003 annual research conference of the South African institute of computer scientists and information technologists on Enablement through technology, South African Institute for Computer Scientists and Information Technologists, 2003, pp. 102–110.

[FJSP93] S. Forrest, B. Javornik, R. E. Smith, and A. S. Perelson, *Using genetic algorithms to explore pattern recognition in the immune system*, Evolutionary Computation **1** (1993), no. 3, 191–211.

[GDH92] David E. Goldberg, Kalyanmoy Deb, and Jeffrey Horn, *Massive multimodality, deception, and genetic algorithms*, Parallel Problem Solving from Nature, 2 (Amsterdam) (R. Männer and B. Manderick, eds.), Elsevier Science Publishers, B. V., 1992.

[Gol89] David E. Goldberg, *Genetic algorithms in search, optimization, and machine learning*, Addison-Wesley, Reading, MA, 1989.

[GR87] David E. Goldberg and Jon Richardson, *Genetic algorithms with sharing for multimodal function optimization*, Proceedings of the Second International Conference on Genetic Algorithms and their application (Mahwah, NJ, USA) (J. J. Grefensette, ed.), Lawrence Erlbaum Associates, Inc., 1987, pp. 41–49.

[GSB02] M. Gunes, U. Sorges, and I. Bouazizi, *Ara-the ant-colony based routing algorithm for manets*, Proceedings of the International Conference on Parallel Processing Workshops, 2002., 2002, pp. 79– 85.

[GV03] V. G. Gudise and G. K. Venayagamoorthy, *Comparison of particle swarm optimization and backpropagation as training algorithms for neural networks*, Proceedings of the IEEE Swarm Intelligence Symposium (SIS '03), April 2003, pp. 110–117.

[Har95] Georges R. Harik, *Finding multimodal solutions using restricted tournament selection*, Proceedings of the Sixth International Conference on Genetic Algorithms (San Francisco, CA) (Larry Eshelman, ed.), Morgan Kaufmann, 1995, pp. 24–31.

[Hay99] S. Haykin, *Neural networks, A comprehensive foundation*, 2nd ed., Prentice Hall, 1999.

[HG90] F. Heppner and U. Grenander, *A stochastic nonlinear model for coordinated bird flocks*, The Ubiquity of Chaos (S. Krusna, ed.), AAAS Publications, Washington, DC, 1990, pp. 233–238.

[Hol75] J. H. Holland, *Adaptation in natural and artificial system*, The University of Michigan Press, Ann Arbor, MI, 1975.

[Hol92] ———, *Adaptation in natural and artificial systems: an introductory analysis with applications to biology*, MIT Press, Cambridge, MA, 1992.

[Hor97] Jeffrey Horn, *The nature of niching: Genetic algorithms and the evolution of optimal, cooperative populations*, Ph.D. thesis, University of Illinois at Urbana Champaign, 1997.

[KE95] J. Kennedy and R. C. Eberhart, *Particle swarm optimization*, Proceedings of the IEEE International Conference on Neural Networks, IV (Piscataway, NJ), IEEE Service Center, 1995, pp. 1942–1948.

[KE97] J. Kennedy and R. C. Eberhart, *A discrete binary version of the particle swarm algorithm*, Proceedings of the World Multiconference on Systemics, Cybernetics, and Informatics (Piscataway, NJ), 1997, pp. 4104–4109.

[KE01] J. Kennedy and R. C. Eberhart, *Swarm intelligence*, Morgan Kaufmann Publishers Inc., 2001.

[Ken97] J. Kennedy, *The particle swarm: Social adaptation of knowledge*, Proceedings of the 1997 International Conference on Evolutionary Computation (Piscataway, NJ), IEEE Service Center, 1997, pp. 303–308.

[Ken99] J. Kennedy, *Small worlds and mega-minds: effects of neighborhood topology on particle swarm performance*, Proceedings of the 1999 Congress on Evolutionary Computation, 1999. CEC 99., vol. 3, 1999, pp. –1938.

[Ken00] ———, *Stereotyping: Improving particle swarm performance with cluster analysis*, Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2000) (San Diego, California, U.S.A.), 2000, pp. 1507–1512.

[KGV83] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, *Optimization by simulated annealing*, Science **220** (1983), no. 4598, 671–680.

[KL02] T. Krink and M. Løvbjerg, *The LifeCycle model: Combining particle swarm optimisation, genetic algorithms and hillclimbers*, Proceedings of Parallel Problem Solving from Nature VII (PPSN-2002), Springer Verlag, 2002, pp. 621–630.

[KM02] J. Kennedy and R. Mendes, *Population structure and particle swarm performance*, Proceedings of the Congress on Evolutionary Computation (CEC '02), vol. 2, 2002, pp. 1671–1676.

[KM03] ———, *Neighborhood topologies in fully-informed and best-of-neighborhood particle swarms*, Proceedings of the 2003 IEEE International Workshop on Soft Computing in Industrial Applications, 2003. SMCia/03., 2003, pp. 45–50.

[Koh01] T. Kohonen, *Self-organizing maps*, 3rd ed., Springer-Verlag, Berlin, 2001.

[Koz92] J. R. Koza, *Genetic programming — on the programming of computers by means of natural selection*, MIT Press, Cambridge, MA, 1992.

[KT01] T. Krink and R. Thomsen, *Self-organized criticality and mass extinction in evolutionary algorithms*, Proceedings of the Third Congress on Evolutionary Computation (CEC-2001) (Seoul, South Korea), vol. 2, 2001, pp. 1155–1161.

[Kun04] Ludmila I. Kuncheva, *Combining pattern classifiers: Methods and algorithms*, Wiley-Interscience, 2004.

[KVR02] T. Krink, J. S. Vesterstrøm, and J. Riget, *Particle swarm optimisation with spatial particle extension*, Proceedings of the Fourth Congress on Evolutionary Computation (CEC-2002) (D. B. Fogel, X. Yao,

G. Greenwood, H. Iba, P. Marrow, and M. Shackleton, eds.), vol. 2, 2002, pp. 1474–1479.

[LBPC02] Jian-Ping Li, Marton E. Balazs, Geoffrey T. Parks, and P. John Clarkson, *A species conserving genetic algorithm for multimodal function optimization*, Evolutionary Computation **10** (2002), no. 3, 207–234.

[Li04] X. Li, *Adaptively choosing neighbourhood bests using species in a particle swarm optimizer for multimodal function optimization*, Proceeding of Genetic and Evolutionary Computation Conference 2004 (GECCO'04) (Seattle, U.S.A.) (K. Deb et al., ed.), Lecture Notes in Computer Science, vol. 3102, Springer-Verlag, 2004, pp. 105–116.

[LK02] M. Løvbjerg and T. Krink, *Extending particle swarms with self-organized criticality*, Proceedings of the Fourth Congress on Evolutionary Computation (CEC-2002) (D. B. Fogel, X. Yao, G. Greenwood, H. Iba, P. Marrow, and M. Shackleton, eds.), vol. 2, 2002, pp. 1588–1593.

[LRK01] M. Løvbjerg, T. Kiel Rasmussen, and T. Krink, *Hybrid particle swarm optimiser with breeding and subpopulations*, Proceedings of the third Genetic and Evolutionary Computation Conference (GECCO-2001), vol. 1, 2001, pp. 469–476.

[Mah95a] S. W. Mahfoud, *Niching methods for genetic algorithms*, Ph.D. thesis, University of Illinois at Urbana-Champaign, 1995.

[Mah95b] Samir W. Mahfoud, *A comparison of parallel and sequential niching methods*, Proceedings of the 6th International Conference on Genetic Algorithms (San Francisco, CA, USA), Morgan Kaufmann Publishers Inc., 1995, pp. 136–143.

[Mil94] M. M. Millonas, *Swarms, phase transitions, and collective intelligence*, Artificial Life III (C.G. Langton, ed.), Addison Wesley, Reading, MA, 1994.

[MKN03] R. Mendes, J. Kennedy, and J. Neves, *Watch thy neighbor or how the swarm can learn from its environment*, Proceedings of the 2003 IEEE Swarm Intelligence Symposium, 2003. SIS '03., 2003, pp. 88–94.

[MP00] G.J. McLachlan and D. Peel, *Finite mixture models*, Wiley series in Probability and Mathematical Statistics: Applied Probability and Statistics Section, no. xvii, John Wiley and Sons, New York, 2000.

[MS96] Brad L. Miller and Michael J. Shaw, *Genetic algorithms with dynamic niche sharing for multimodal function optimization*, International Conference on Evolutionary Computation, 1996, pp. 786–791.

[OGC91] C. Oei, D. E. Goldberg, and S. Chang, *Tournament selection, niching, and the preservation of diversity*, IlliGAL Report 91011, University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory, Urbana, 1991.

[OM98] E. Ozcan and C. Mohan, *Analysis of a simple particle swarm optimization system*, Intell. Eng. Syst. Through Artif. Neural Networks **8** (1998), 253–258.

[OM99] E. Ozcan and C. K. Mohan, *Particle swarm optimization: surfing the waves*, Proceedings of the Congress on Evolutionary Computation (CEC 99), vol. 3, 1999, pp. 1939–1944.

[PE03] U. Paquet and A.P. Engelbrecht, *Training support vector machines with particle swarms*, Proceedings of the International Joint Conference on Neural Networks, vol. 2, July 2003, pp. 1593–1598.

[Pet96] A. Petrowski, *A clearing procedure as a niching method for genetic algorithms*, Proceedings of IEEE International Conference on Evolutionary Computation, 1996.

[PL04] D. Parrott and X. Li, *A particle swarm model for tracking multiple peaks in a dynamic environment using speciation*, Proceeding of the 2004 Congress on Evolutionary Computation (CEC'04), IEEE Service Center, 2004, pp. 98–103.

[PM00] Dan Pelleg and Andrew Moore, *X-means: Extending k-means with efficient estimation of the number of clusters*, Proceedings of the

Seventeenth International Conference on Machine Learning (San Francisco), Morgan Kaufmann, 2000, pp. 727–734.

[PPMV01a] K.E. Parsopoulos, V.P. Plagianakos, G.D. Magoulas, and M.N. Vrahatis, *Improving particle swarm optimizer by function "stretching"*, Nonconvex Optimization and Applications, vol. 54, ch. 3, pp. 445–457, Kluwer Academic Publishers, The Netherlands, 2001.

[PPMV01b] ———, *Stretching technique for obtaining global minimizers through particle swarm optimization*, Proceedings of the Particle Swarm Optimization Workshop (Indianapolis (IN), U.S.A.), 2001, pp. 22–29.

[PS06] A. Passaro and A. Starita, *Clustering particles for multimodal function optimization*, Proceedings of EC^{2}AI - ECAI Workshop on Evolutionary Computation (Riva del Garda, Italy), 2006.

[PV01] K.E. Parsopoulos and M.N. Vrahatis, *Modification of the particle swarm optimizer for locating all the global minima*, Artificial Neural Networks and Genetic Algorithms (V. Kurkova, N.C. Steele, R. Neruda, and M. Karny, eds.), Computer Science series, Springer, Wien, 2001, pp. 324–327.

[PV04] K.E. Parsopoulos and M. N. Vrahatis, *On the computation of all global minimizers through particle swarm optimization*, IEEE transactions on evolutionary computation **8** (2004), no. 3, 211–224.

[PVM03] T. Peram, K. Veeramachaneni, and C. K. Mohan, *Fitness-distance-ratio based particle swarm optimization*, Proceedings of the IEEE Swarm Intelligence Symposium (SIS 2003) (Indianapolis, Indiana, USA), 2003, pp. 174–181.

[Ree83] W. T. Reeves, *Particle systems - a tecnique for modeling a class of fuzzy objects*, ACM Transactions on Graphics **2** (1983), no. 2, 91–108.

[Rey87] C. W. Reynolds, *Flocks, herds, and schools: A distributed behavioral model, in computer graphics*, Proceedings of the SIGGRAPH '87 Conference, 1987, pp. 25–34.

[RHW04] A. Ratnaweera, S.K. Halgamuge, and H.C. Watson, *Self-organizing hierarchical particle swarm optimizer with time-varying acceleration*

coefficients, IEEE Transactions on Evolutionary Computation **8** (2004), no. 3, 240–255.

[RN95] Stuart J. Russell and Peter Norvig, *Artificial intelligence: a modern approach*, Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1995.

[Sch78] Gideon Schwarz, *Estimating the dimension of a model*, The Annals of Statistics **6** (1978), no. 2, 461–464.

[SE98a] Y. Shi and R. C. Eberhart, *A modified particle swarm optimizer*, The 1998 IEEE International Conference on Evolutionary Computation Proceedings (Anchorage, AK), May 1998, pp. 69–73.

[SE98b] Y. Shi and R. C. Eberhart, *Parameter selection in particle swarm optimization*, EP '98: Proceedings of the 7th International Conference on Evolutionary Programming VII, Springer-Verlag, 1998, pp. 591–600.

[SE04] I.L. Schoeman and A.P. Engelbrecht, *Using vector operations to identify niches for particle swarm optimization*, Proceedings of the IEEE 2004 Conference on Cybernetics and Intelligent Systems, 2004, pp. 361–366.

[SE05] ———, *A parallel vector-based particle swarm optimizer*, International Conference on Artificial Neural Networks and Genetic Algorithms (ICANNGA 2005) (Portugal), 2005.

[SG05] J. F. Schutte and A. A. Groenwold, *A study of global optimization using particle swarms*, Journal of Global Optimization **31** (2005), no. 1, 93–108.

[SK98] B. Sareni and L. Krahenbuhl, *Fitness sharing and niching methods revisited*, IEEE Transactions on Evolutionary Computation **2** (1998), no. 3, 97–106.

[SSN03] T. Sousa, A. Silva, and A. Neves, *A particle swarm data miner.*, Progress in Artificial Intelligence, 11th Protuguese Conference on Artificial Intelligence, EPIA, 2003, pp. 43–53.

[Sug99] P.N. Suganthan, *Particle swarm optimiser with neighbourhood operator*, Proceedings of the Congress on Evolutionary Computation (CEC 99), vol. 3, 1999, pp. 1959–1962.

[Tre03] I. C. Trelea, *The particle swarm optimization algorithm: convergence analysis and parameter selection*, Inf. Process. Lett. **85** (2003), no. 6, 317–325.

[vdB02] Frans van den Bergh, *An analysis of particle swarm optimizers*, Ph.D. thesis, Department of Computer Science, University of Pretoria, Pretoria, South Africa, 2002, Supervisor-A. P. Engelbrecht.

[vdBE02] F. van den Bergh and A. P. Engelbrecht, *A new locally convergent particle swarm optimizer*, IEEE Conference on Systems, Man, and Cybernetics, 2002.

[vN51] John von Neumann, *The general and logical theory of automata*, Cerebral Mechanism in Behavior (New York) (L.A. Jeffress, ed.), John Wiley and Sons, 1951.

[vNB66] John von Neumann and Arthur W. Burks, *Theory of self-reproducing automata*, University of Illinois Press, Champaign, IL, USA, 1966.

[VPMO03] K. Veeramachaneni, T. Peram, C. K. Mohan, and L. A. Osadciw, *Optimization using particle swarms with near neighbor interactions.*, Proceeding of Genetic and Evolutionary Computation Conference 2003 (GECCO'03), 2003, pp. 110–121.

[VRK02] J. S. Vesterstrøm, J. Riget, and T. Krink, *Division of labor in particle swarm optimisation*, Proceedings of the Fourth Congress on Evolutionary Computation (CEC-2002) (D. B. Fogel, X. Yao, G. Greenwood, H. Iba, P. Marrow, and M. Shackleton, eds.), vol. 2, 2002, pp. 1570–1575.

[WC92] Michael P. Windham and Adele Cutler, *Information ratios for validating mixture analyses*, Journal of the American Statistical Association **87** (1992), no. 420, 1188–1192.

[Wil75] E. O. Wilson, *Sociobiology: The new synthesis*, Belknap Press, Cambridge, MA, 1975.

[WS98] Duncan J. Watts and Steven H. Strogatz, *Collective dynamics of 'small-world' networks*, Nature **393** (1998), no. 6684, 440.

[XDE+03] X. Xiao, E. R. Dow, R. C. Eberhart, Z. B. Miled, and R. J. Oppelt, *Gene clustering using self-organizing maps and particle swarm optimization*, IPDPS '03: Proceedings of the 17th International Symposium on Parallel and Distributed Processing, IEEE Computer Society, 2003, pp. 154–163.

[XW05] R. Xu and D. Wunsch, *Survey of clustering algorithms*, IEEE Transactions on Neural Networks **3** (2005), no. 16, 645–678.

[YG93] X. Yin and N. Germay, *A fast genetic algorithm with sharing scheme using cluster methods in multimodal function optimization*, Proceedings of the International Conference on Artificial Neural Nets and Genetic Algorithms (R. F. Albrecht, C. R. Reeves, and N. C. Steele, eds.), Springer-Verlag, 1993, pp. 450–457.

[YKFN99] H. Yoshida, K. Kawata, Y. Fukuyama, and Y. Nakanishi, *A particle swarm optimization for reactive power and voltage control considering voltage stability*, Proceedings of the International Conference on Intelligent System Application to Power Systems (Rio de Janeiro, Brazil) (G. L. Torres and A. P. Alves da Silva, eds.), 1999, pp. 117–121.

[ZMZQ03] Y. L. Zheng, L. H. Ma, L. Y. Zhang, and J. X. Qian, *On the convergence analysis and parameter selection in particle swarm optimization*, International Conference on Machine Learning and Cybernetics, vol. 3, Nov. 2003, pp. 1802–1807.