Université Libre de Bruxelles
Faculté des Sciences Appliquées
IRIDIA - *Institut de Recherches Interdisciplinaires et de Développements en Intelligence Artificielle*

On the Performance of Particle Swarm Optimizers

par

Marco A. Montes de Oca Roldán

Directeur de Mémoire:

Prof. Marco Dorigo

Co-promoteur de Mémoire:

Dr. Thomas Stützle

Mémoire présenté en vue de l'obtention du Diplôme d'Études Approfondies en Sciences Appliquées

Année Académique 2005/2006

Abstract

Since the introduction of the first Particle Swarm Optimization algorithm by James Kennedy and Russell Eberhart in the mid-90's, many variants of the original algorithm have been proposed. However, there is no general agreement on which variant(s) could be considered the state-of-the-art in the field. This is, in part, due to a general lack of cross-comparisons among variants in the literature.

The work reported in this document was carried out with the goal of identifying the best-performing particle swarm optimization algorithms. For practical reasons, we could not compare all the available algorithmic variants. Instead, we focused on those that have been the most widely used variants. We also considered algorithms that incorporate some of the latest developments in field.

The comparison of the chosen particle swarm optimization algorithms was based on run-length and solution-quality distributions. These analytical tools allow the comparison of stochastic optimization algorithms in terms of the probability of finding a solution to an optimization problem within some minimum acceptable solution quality and time bounds. This kind of analysis is helpful for measuring the solution-quality vs. computational effort tradeoff each algorithm presents.

Particle swarm optimizers are population-based optimization algorithms that rely on a population structure referred to as *topology*. We conduct an analysis of the effects that different population topologies (the most commonly used ones) have in their performance. We show how some topologies are better suited for application scenarios in which only short runs are allowed or affordable, and how other topologies are better suited for situations in which the only important factor is solution quality.

Run-length and solution-quality distributions also serve as diagnostic tools that help in the design of improved algorithms. After analyzing the behavior of some variants with these tools, we found opportunities for improvement and the resultant variants are presented and discussed.

Acknowledgments

Even though theses are attributed to a single person, most of them are the result of many hours of discussion between different people. This thesis is not an exception. As the author of this thesis, I have been lucky for working at IRIDIA (*Institut de Recherches Interdisciplinaires et de Devéloppments en Intelligence Artificielle*) which is part of the *Université Libre de Bruxelles* and being able to interact with all the brilliant people that work there.

I would like to express my gratitude to Dr. Thomas Stützle for all his critical reviews of this work and his constant encouragement. I am also grateful to Dr. Mauro Birattari for his constructive suggestions for the improvement of this document. Finally, I would like to thank Prof. Marco Dorigo, my supervisor, for his suggestions, ideas (in fact, this work is derived from one of them) and for the opportunity of coming to Brussels in the first place.

This work has been supported by the ANTS project, an *Action de Recherche Concertée* funded by the Scientific Research Directorate of the French Community of Belgium, and by the Programme Alβan, the European Union Programme of High Level Scholarships for Latin America, through scholarship No. E05D054889MX.

August, 2006.

Marco A. MONTES DE OCA ROLDAN

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

The introduction of the first Particle Swarm Optimization (PSO) algorithm by Kennedy and Eberhart [23] [11], gave birth to a field with the same name. Since then, it has received a growing interest from the industrial and academic worlds.

It has been already eleven years since the birth of the Particle Swarm Optimization field and during this time, tens of algorithmic variants have been proposed. For years, authors have been comparing their variants only with one particle swarm optimization algorithm. In most cases, as expected, they "outperformed" the reference algorithm. We believe that this hinders the true advancement of the field: how can we build on previous efforts if the state-of-the-art is not well identified? Surprisingly, there are no comparisons among particle swarm optimization algorithms in the current literature.

1.1 Objective

The main objective of this work is to identify the algorithmic variants that can be considered as the state-of-the-art in the field.

To fulfill this objective, we carried out a comparison among the most widely used particle swarm optimization algorithms. We identify the best performing algorithms taking into account the possible application scenarios in which the algorithms may be used. In other words, we identify the circumstances under which the top performing algorithms work at their best. This gives the possibility of choosing different algorithms given different application requirements.

1.2 Methodology

The empirical methodology that we used to perform the comparison is the one proposed by Hoos and Stützle [17]. It is based on the estimation of the run-length and solution-quality distributions exhibited by stochastic optimization algorithms when solving specific problems.

The run-time distribution is the cumulative distribution function of the random variable describing the time needed by a stochastic optimization algorithm to find a solution of a specific quality. Likewise, the solution-quality distribution is the cumulative distribution function of the random variable that represents the solution quality achieved by an algorithm exactly after some computational effort limit.

Run-time and solution-quality distributions provide information about the behavior of stochastic optimization algorithms that is useful for making comparisons and for devising improvements. Specifically, they can tell us how prone an algorithm is to stagnating. An algorithm with propensity to stagnate will show a slowly-increasing or a non-increasing probability (towards the right tail of the distribution) of finding a solution of a given quality over the allocated time limit.

The estimation of run-time and solution quality distributions does not impose a major burden in the process of collecting data. All we need to do is to run the algorithm several times and record information relative to solution improvement. In every run, we need to record the time or the number of critical operations, and the solution quality whenever a new best solution is found by the algorithm.

1.3 Contributions

To the best of our knowledge, this document is the first one in the literature that reports a comparison among the most widely used particle swarm optimization algorithms. A detailed analysis of the performance of each algorithm, allowed us to propose two improved variants.

The first variant is one that incorporates an adaptive restart mechanism. It is based on the fact that, in some cases, many short runs are better than a single long run of a stochastic optimization algorithm [17]. The second variant borrows some ideas from recent developments in Ant Colony Optimization [9]. It is a particle swarm optimizer that uses the information gathered throughout the optimization process, to guide the search into the most promising regions of the search space.

1.3.1 Publications

Part of the material presented in Chapter 4 will be published in:

- M. A. Montes de Oca, T. Stützle, M. Birattari, and M. Dorigo (2006), A Comparison of Particle Swarm Optimization Algorithms Based on Run-Length Distributions. *Proceedings of the Fifth International Workshop on Ant Colony Optimization and Swarm Intelligence ANTS 2006*, Brussels, Belgium. M. Dorigo, L. M. Gambardella, M. Birattari, A. Martinoli, R. Poli, and T. Stützle (Eds.), LNCS 4150. Springer, Berlin, Germany.

Additionally, part of the material presented in Chapter 5 will also be published in:

- M. Iqbal and M. A. Montes de Oca (2006), An Estimation of Distribution Particle Swarm Optimization Algorithm. *Proceedings of the Fifth International Workshop on Ant Colony Optimization and Swarm Intelligence ANTS 2006*, Brussels, Belgium. M. Dorigo, L. M. Gambardella, M. Birattari, A. Martinoli, R. Poli, and T. Stützle (Eds.), LNCS 4150. Springer, Berlin, Germany.

1.4 Structure

This document is organized as follows: in Chapter 2, we present a detailed description of the Particle Swarm Optimization field describing, also, the specific algorithmic variants included in our comparison; in Chapter 3, we describe the empirical evaluation methodology we used to assess the performance of the different particle swarm optimizers included in the comparison; in Chapter 4 we present the comparison among the evaluated algorithms and an analysis of different inertia weight schedules in the particle swarm optimizer with time-decreasing inertia weight; in Chapter 5, we present two improved variants that build on what we learned from the analysis carried out in Chapter 4; finally, in Chapter 6, we summarize the main points discussed in the body of this thesis.

Chapter 2

Particle Swarm Optimization

In this chapter, a detailed description of the Particle Swarm Optimization (PSO) field is presented. In order to have a clearer picture of its development, we first present some of its antecedents and the theories that served as inspiration for its first proponents. We then describe the first algorithmic variants. We continue by presenting the algorithmic variants that are part of our empirical comparison (to be presented in Chapters 4 and 5). Finally, we briefly describe some of the advances done in other aspects of the technique.

2.1 Antecedents

PSO has its roots in computer graphics, simulation and theories of social adaptation. In the following, we describe some of the ideas drawn from these areas that gave birth to PSO.

2.1.1 Particle Systems

In the early 1980's, tridimensional computer-generated graphics started being introduced into TV and film productions. Rigid objects were, and are still, modeled by sets of polygons arranged in such a way as to give the impression of depth. Many objects can be represented in this way, but many others cannot. Examples of such objects are fire, smoke, water and clouds.

To tackle this problem, Reeves [43] proposed particle systems to model "fuzzy" dynamic objects. A particle system is composed of several tiny elements that move in space. These elements have a set of attributes such as an initial position, an initial velocity, a lifetime, an initial size, an initial color, etc. Their kinematics is governed by classical physics laws. A set of rules determine other aspects of the dynamics of a particle system; for example, its color or the size of the particles.

During simulation, each particle is independent of all other particles and moves in a (commonly) three dimensional space. Particles' trajectories are ultimately the result of the interaction of their initial attributes and the environment physics.

2.1.2 Simulated Bird Flocks

Particle systems can exhibit a complex behavior if a script handles each particle's movement or embedded behavior (e.g., color change). Scripts can also take into account some conditions present in the environment. In this way, the behavior of particles can be reactive.

Reynolds [44] used a particle system to simulate the collective behavior of a flock of birds using the reactive script technique mentioned above. In his simulations, particles (representing birds) had a "body" made of a polygon mesh. Static rigid objects were also part of the simulated environment. The novelty in Reynolds' work was that it considered particles as part of the environment. Since the behavior of a particle was controlled by a script and each particle was independent from the others, the interaction between particles gave as a result an "emergent" collective behavior that resembled a flock of birds. The behavioral rules found by Reynolds served as a starting point for the design of PSO.

In Reynolds' model there are three categories of behavioral rules. In order of decreasing precedence, they are:

- Collision Avoidance. These allow a bird-oid (boid) avoid collisions with their close neighbors.
- Velocity Matching. Boids try to keep up with their neighbors.
- Flock Centering. These rules maintain the flock aggregated.

The original PSO algorithm is based on the last two categories since particles do not have volume and therefore, there is no need to take care about collisions. Recent work considers the first category to avoid the group of particles to collapse. We will later describe this idea in more detail.

2.1.3 Social Adaptation

A key hypothesis in the development of the first PSO algorithm is that the exchange of information among members of the same species provides an evolutionary advantage [23]. Examples of phenomena that motivate this hypothesis are fish schooling, bird flocking, animal herding and human social behavior.

Kennedy [21, 22] argues that the search strategy followed by particles in PSO obeys the principles, we humans, use to improve our own performance when solving a problem cooperatively. In particular, the idea that we humans tend to imitate the behavior of others we observe are rewarded in some way. He affirms that in the original PSO algorithm, particles don't just try to imitate the behavior of other particles. They have their own beliefs and they also tend to stick with the best ones. Social adaptation happens when a particle perceives that other particle's beliefs are better than its own.

2.2 The Original Particle Swarm Optimization Algorithm

The PSO concept is a result of mixing the ideas presented in the last section. In a PSO algorithm a number of solutions to an optimization problem are updated according to some form of interaction among them. This is very similar to what happens in a particle system in a computer graphics environment, in which particles react to what other particles do. The rules devised by Reynolds to simulate a flock of birds inspired the rules that governs the update rules in a PSO algorithm. Having a rough historical perspective of the development of the PSO technique, let us now go into its details.

In order to optimize an unconstrained d-dimensional objective function $f : \mathbb{R}^d \to \mathbb{R}$, the original PSO algorithm [11, 23] is initialized with a population of complete solutions (called particles) $\{p_1, \ldots, p_k\} = \mathcal{P}$, randomly located in the solution space. The objective function determines the quality of a particle's position, that is, the quality of the solution it represents.

A particle p_i at time step t has a position vector $\vec{x}_i^t$ and an associated velocity vector $\vec{v}_i^t$. Every particle "remembers" the position in which it has received the best evaluation of the objective function. This memory is represented by vector $\vec{pbest}_i$. This vector is updated every time particle p_i finds a better position.

At the swarm level, the vector $\vec{gbest}$ stores the best position any particle has ever visited. This means that if we face a minimization problem, then $f(\vec{gbest}) \leq f(\vec{pbest}_i)\, \forall p_i \in \mathcal{P}$. A similar idea is used for maximization problems.

The algorithm iterates updating particles' velocity and position until a stopping criterion is met, usually a sufficiently good solution value or a maximum number of iterations or function evaluations. The update rules are:

$$\vec{v}_i^{t+1} = \vec{v}_i^t + \varphi_1 \cdot \vec{U}_1(0,1) * (\vec{pbest}_i - \vec{x}_i^t) + \varphi_2 \cdot \vec{U}_2(0,1) * (\vec{gbest} - \vec{x}_i^t)\,, \tag{2.1}$$

$$\vec{x}_i^{t+1} = \vec{x}_i^t + \vec{v}_i^{t+1}\,, \tag{2.2}$$

where φ_1 and φ_2 are two constants called the *cognitive* and *social* coefficients respectively, $\vec{U}_1(0,1)$ and $\vec{U}_2(0,1)$ are two d-dimensional uniformly distributed random vectors (generated every iteration) in which each component goes from zero to one, and $*$ is an element-by-element vector multiplication operator. The values of φ_1 and φ_2 are parameters of the algorithm, but in the original version they are given a value of 2.

In Algorithm 1 we show a pseudocode version of the original PSO algorithm. Of course, in a real implementation, every time a particle is evaluated, the value returned by the objective function is stored in variables. In Algorithm 1 this level of detail is not present.

In the original PSO algorithm a particle has two attractors: its own previous best position, and the swarm's global best position. Previous experience with population-based optimization algorithms dictated that a strong bias towards the best solution so far may lead to premature convergence; therefore, the so-called local version of the PSO algorithm was devised.

2.2.1 Local Particle Swarm Optimizer

An early variant of the original PSO algorithm was proposed by Eberhart and Kennedy [11] in which a particle does not accelerate towards the swarm's global best solution. Instead, it accelerates towards the best solution found within its local *topological* neighborhood. A particle p_i has a topological neighborhood $\mathcal{N}_i \subseteq \mathcal{P}$ (remember that $\mathcal{P}$ is the set of particles in the swarm) of particles. Note that this definition does not restrict a particle p_i of being a neighbor of

Algorithm 1 Pseudocode version of the original PSO algorithm

```
{Initialization}
for $i = 1$ to $k$ do
  Create particle $p_i$
  Initialize its vectors $\vec{x}_i$ and $\vec{v}_i$ to random values
  Set $\vec{pbest}_i = \vec{x}_i$
end for
Set $\vec{gbest} = \vec{x}_0$

{Main Loop}
Set $t = 0$
while $gbest$ is not good enough or $t < t_{max}$ do
  {Evaluation Loop}
  for $i = 1$ to $k$ do
    if $f(\vec{x}_i)$ is better than $f(\vec{pbest}_i)$ then
      Set $\vec{pbest}_i = \vec{x}_i$
    end if
    if $f(\vec{pbest}_i)$ is better than $f(\vec{gbest})$ then
      Set $\vec{gbest} = \vec{pbest}_i$
    end if
  end for
  {Update Loop}
  for $i = 1$ to $k$ do
    Generate $\vec{U}_1(0,1)$ and $\vec{U}_2(0,1)$
    Set $\vec{v}_i = \vec{v}_i + \varphi_1 \cdot \vec{U}_1(0,1) * (\vec{pbest}_i - \vec{x}_i) + \varphi_2 \cdot \vec{U}_2(0,1) * (\vec{gbest} - \vec{x}_i)$
    Set $\vec{x}_i = \vec{x}_i + \vec{v}_i$
  end for
  Set $t = t + 1$
end while
```

itself. It is called *topological*, because it does not consider the actual spatial arrangement of the particles in the search space.

In the local version of the PSO algorithm, Equation 2.1 becomes

$$\vec{v}_i^{t+1} = \vec{v}_i^t + \varphi_1 \cdot \vec{U}_1(0,1) * (\vec{pbest}_i - \vec{x}_i^t) + \varphi_2 \cdot \vec{U}_2(0,1) * (\vec{lbest}_i - \vec{x}_i^t)\,, \quad (2.3)$$

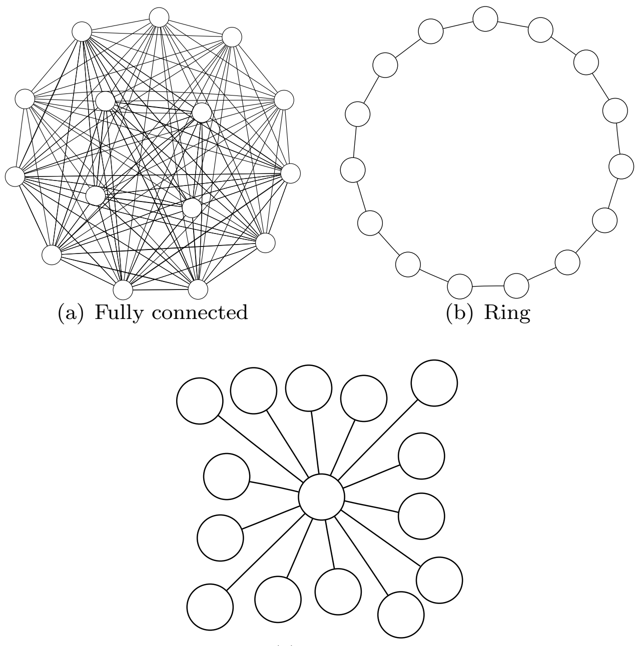
where $\vec{lbest}_i = \underset{p_j \in \mathcal{N}_i}{\operatorname{argmin}} f(\vec{pbest}_j)$ for minimization problems[1]. Note that the original version of the PSO is just a special case of the local version using a fully connected topology.

The definition of the topological neighborhood is dependent on the way the algorithm is implemented. In the original proposal, all particles were assumed to be stored in a cyclic bidimensional array. To determine the vector $\vec{lbest}_i$, particle p_i compares its solution against those of particles stored in indices $i-m$ to $i+m$, with m being the so-called "radius" of the neighborhood.

The topological arrangement just described, is known as the circle or ring topology, but there are others such as the wheel topology in which all individuals

[1]From now on, all descriptions will assume minimization problems. Similar reasoning can be used for maximization problems.

are connected to a single central particle. Figure 2.1 shows some topological arrangements used in PSO.


(a) Fully connected (b) Ring

(c) Wheel

Figure 2.1: Some commonly used topologies in PSO algorithms. Nodes represent particles and edges represent inter-particle influences. A particle's neighborhood is composed of all the particles reachable by an edge.

2.3 Particle Swarm Optimizers

One of the most active areas of research in PSO has been that of algorithmic modifications. This has resulted in several algorithmic variations of the original PSO algorithm. In the following subsections we will describe the algorithmic variants that are part of our empirical comparison. In our eyes, they are among the most influential and promising algorithmic variations. The order of description is (to some extent) chronological. This will help the reader to appreciate the significance of some of the ideas introduced by each variant.

2.3.1 Canonical Particle Swarm Optimizer

Clerc and Kennedy [7] introduced a constriction factor into the velocity update rule of the original PSO algorithm. The purpose of this factor is to avoid particles' velocities to increase towards infinity, a phenomenon present in the original PSO [12, 49].

This constriction factor is added in Equation 2.3 giving

$$\vec{v}_i^{t+1} = \chi \cdot \left(\vec{v}_i^t + \varphi_1 \cdot \vec{U}_1(0,1) * (\vec{pbest}_i - \vec{x}_i^t) + \varphi_2 \cdot \vec{U}_2(0,1) * (\vec{lbest}_i - \vec{x}_i^t)\right) \,, \tag{2.4}$$

with

$$\chi = \frac{2 \cdot k}{\left|2 - \varphi - \sqrt{\varphi^2 - 4\varphi}\right|} \,, \tag{2.5}$$

where $k \in [0,1]$, $\varphi = \varphi_1 + \varphi_2$ and $\varphi > 4$. Usually, k is set to 1 and both φ_1 and φ_2 are set to 2.05, giving as a result χ equal to 0.729 [12, 56].

Even though Clerc and Kennedy proposed a way to choose some of the free parameters of the original PSO and not really a different algorithm, it has been so widely used that it is known as the *canonical* PSO.

2.3.2 Time-Varying Inertia Weight Particle Swarm Optimizer

Shi and Eberhart [48, 50] noticed that by eliminating the first term of Equation 2.1, the particle swarm searched within the region around particles' $\vec{lbest}_i$ vectors; hence, they concluded that the velocity term gives the swarm the opportunity to explore other regions.

Given the fact that it is widely accepted that an optimization algorithm should balance its diversification–intensification behavior, Shi and Eberhart introduced the idea of a control factor called *inertia weight.* The velocity update rule was modified as follows

$$\vec{v}_i^{t+1} = w(t) \cdot \vec{v}_i^t + \varphi_1 \cdot \vec{U}_1(0,1) * (\vec{pbest}_i - \vec{x}_i^t) + \varphi_2 \cdot \vec{U}_2(0,1) * (\vec{lbest}_i - \vec{x}_i^t) \,, \tag{2.6}$$

where $w(t)$ is the inertia weight which is usually a time-dependent function. In this PSO variant, φ_1 and φ_2 are both set to 2 just as in the original PSO algorithm. Note that the canonical PSO is a special case of the inertia weight variant in which $w(t) = 0.729$ and the coefficients φ_1 and φ_2 are set to $0.729 \cdot 2.05 = 1.49445$.

Intuition dictates that $w(t)$ should be a time-decreasing function of time since we want the algorithm to explore the search space during the first iterations and to focus on the promising regions afterwards. This approach was the one proposed by Shi and Eberhart and was proved to be useful [48, 49, 50]. Nevertheless, Zheng et al. [58, 59] studied the effects of using a time-increasing inertia weight function showing also that, in some cases, it provides a faster convergence rate. An investigation on the nonlinearity of a time-decreasing inertia weight function was conducted by Chatterjee and Siarry [6].

The function used to schedule the inertia weight is defined as

$$w(t) = \frac{t_{max} - t}{t_{max}} \cdot (inertia_{max} - inertia_{min}) + inertia_{max} \,, \tag{2.7}$$

where t_{max} marks the time at which $w(t) = inertia_{min}$, $inertia_{max}$ and $inertia_{min}$ are the maximum and minimum values the inertia weight can take, respectively.

The most widely used approach, is the one that uses a decreasing inertia weight with a starting value of 0.9 and 0.4 as the final one. Zheng et al., use the opposite settings. In this case, Equation 2.7 is used in the same way, except that $inertia_{max}$ and $inertia_{min}$ are interchanged.

2.3.3 Stochastic Inertia Weight Particle Swarm Optimizer

Eberhart and Shi [13] proposed another variant in which the inertia weight is randomly selected according to a uniform distribution in the range [0.5,1.0]. This range was inspired by Clerc and Kennedy's constriction factor because the expected value of the inertia weight in this case is $0.75 \approx 0.729$. In this version, the acceleration coefficients are set to 1.494 as a result of the multiplication $\chi \cdot \varphi_{1,2}$. Although this variant was originally proposed for dynamic environments, it has also been shown to be a competitive optimizer for static ones [42].

2.3.4 Fully Informed Particle Swarm Optimizer

In PSO, population topologies serve as a method to select the vector $\vec{lbest}_i$ that will guide particle i in its search. The function of vector $\vec{lbest}_i$ is that of an attractor. But the attraction point need not be a position previously visited by a particle. In the spirit of the PSO paradigm, it just needs to be a "good" (in terms of the value of the objective function) point.

Based on this reasoning, Mendes et al. [33] proposed the fully informed particle swarm (FIPS), in which a particle uses information from all its topological neighbors. This variant also uses Clerc and Kennedy's constriction factor since it does not enforce that the value φ (see Section 2.3.1) should be split only between two attractors.

For a given particle, the way φ (i.e., the sum of the acceleration coefficients) is decomposed is $\varphi_k = \varphi/|\mathcal{N}_i| \ \forall p_k \in \mathcal{N}_i$ where $\mathcal{N}_i$ is the neighborhood of particle i. As a result, the new velocity update equation becomes

$$\vec{v}_i^{\,t+1} = \chi \left[\vec{v}_i^{\,t} + \sum_{p_k \in \mathcal{N}_i} \varphi_k \cdot \mathcal{W}(\vec{pbest}_k) \cdot \vec{U}_k(0,1) * (\vec{pbest}_k - \vec{x}_i^{\,t}) \right] , \qquad (2.8)$$

where $\mathcal{W} : \mathbb{R}^d \to \mathbb{R}$ is a weighting function. The goal of $\mathcal{W}(\vec{pbest}_i)$ is to provide information about the quality of the attractor $\vec{pbest}_i$. The normalized objective function value of vector $\vec{pbest}_i$ could serve well, for example.

The effects of using different topologies on the performance and behavior of PSO algorithms are presented in Section 2.4.2.

2.3.5 Self-Organizing Hierarchical Particle Swarm Optimizer with Time-varying Acceleration Coefficients

We mentioned in Section 2.3.2 that Shi and Eberhart noticed a convergent behavior of the particle swarm if the previous velocity term was eliminated from the velocity update rule. Ratnaweera et al. [42], contrary to the approach taken by Shi and Eberhart, kept this modification and exploited it.

In HPSOTVAC, if any component of a particle's velocity vector becomes zero, it is reinitialized to a value proportional to the maximum allowable velocity V_{max}. To amplify the local search behavior of the swarm, HPSOTVAC linearly adapts the value of the acceleration coefficients φ_1 and φ_2. The cognitive coefficient, φ_1, is decreased from 2.5 to 0.5 and the social coefficient, φ_2, is increased from 0.5 to 2.5.

To avoid the problem of setting a proper reinitialization velocity, HPSOTVAC linearly decreases it from V_{max} at the beginning of the run to $0.1 \cdot V_{max}$ at the end. As in the time-decreasing inertia weight variant, a low reinitialization velocity near the end of the run, allows particles to move slowly near the best region they found.

2.3.6 Adaptive Hierarchical Particle Swarm Optimizer

Proposed by Janson and Middendorf [19], the adaptive hierarchical PSO (AH-PSO) is an example of a PSO with dynamic adaptation of the population topology. Contrary to other approaches, such as the one of Suganthan [54] in which the Euclidean distance between particles was used to decide when to add (or delete) a particle from another particle's neighborhood, Janson and Middendorf do not use a time-intensive operation as a criterion to adapt the topology.

In addition, in AHPSO, the topology is a tree-like structure in which particles with better objective function evaluations are located in the upper nodes of the tree. At each iteration, a child particle updates its velocity considering its own previous best performance and the previous best performance of its parent. Additionally, before the velocity updating process takes place, the previous best fitness value of any particle is compared with that of its parent. If it is better, child and parent swap their positions in the hierarchy.

The branching degree of the tree is a factor that can balance the diversification-intensification behavior of the algorithm. To dynamically adapt the algorithm to the stage of the optimization process, the branching degree is decreased by k_{adapt} degrees until a certain minimum degree d_{min} is reached. This process takes place every f_{adapt} number of iterations.

2.4 Some Advances in Other Aspects of Particle Swarm Optimization

Research in PSO has not only focused on the development of algorithmic variants but also on other fundamental aspects. For example, much effort has been devoted to the problem of choosing an optimal set of parameters for PSO algorithms. There has also been research on the effects of using different population topologies. There have been hybridizations with other optimization techniques and many applications have been proposed.

In this section, we will briefly describe some of the most important results that have been obtained in these areas.

2.4.1 Convergence Analysis and Parameter Selection

Since its birth, PSO research was of an empirical nature. There was no informed way to tune parameters and they were usually set after some trial and error tests. Moreover, there was no clue about how the algorithm actually worked or if it was able to converge at all.

Early on, it was clear that (sometimes) particles increased their velocities very fast. This caused the particle swarm to "explode" [7]. The first approach to cope with this problem was to clamp the maximum allowable velocity to a predefined value. Empirical evidence suggested that a good choice was to set $V_{max} = X_{max}$ where X_{max} is the limit of the search range [49].

The problem of increasing velocities is that particles may go outside the search range and never come back again. If a particle swarm explodes, there is no hope of solving the problem at hand. This motivated researchers to look closer at particles' behavior.

The first efforts to formally analyze the behavior of a particle in a search space were conducted by Ozcan and Mohan [36, 37]. Their study was focused on a simplified version of the original version of the PSO algorithm. They found that the behavior of particles is strongly governed by the value $\varphi = \varphi_1 + \varphi_2$. Specifically, they found that depending on the value this parameter could take, a particle will move over the optimum following a sinusoidal trajectory with different amplitudes and frequencies. In one particular case, the amplitude of a particle's sinusoidal trajectory increases, drifting it to the infinity.

We already mentioned Clerc and Kennedy's work [7] on PSO convergence analysis and their addition of the constriction factor to the classical velocity update equation. A detailed exposition of the mathematical process through which Equations 2.4 and 2.5 were derived is not presented here. The interested reader is referred to the original article. It is important to note, however, that the use of the word *convergence* in Clerc and Kennedy's work can be misleading for those readers who are familiar with optimization theory. The constriction factor forces the swarm to converge to a stable point in the search space which might or might not be a local optimum. It a mechanism intended to avoid the explosion of the swarm.

Trelea [56] presented another approach to determine convergent behavior and gave some guidelines to the selection of the constriction factor and the acceleration coefficients.

The formal analyses carried out up to now, have considered a simplified particle swarm model in which randomness is eliminated and there is no interaction between particles. As a result, even though there are some interesting theoretical results, empirical work is still needed to tune the parameters of a PSO algorithm for solving a particular problem.

Poli et al. [39, 40] applied genetic programming techniques to evolve the velocity and position update rules of a particle swarm when solving specific problems. This work can be considered the "extreme" with respect to algorithm tuning since, in fact, specific variants are devised for specific problems.

2.4.2 Population Topologies

The behavior of a particle swarm is the result of the trajectories followed by particles through the search space, which in turn are the result of particles' past success and that of their neighbors. For an adequate performance, it is essential to properly select particles' neighbors. Instead of choosing a neighbor based on its position in space (which implies the use of costly procedures, such as the computation of the Euclidean distance), a mechanism that permits a cheap neighbor selection is that of a population topology.

The early proposal of a local version of the original PSO algorithm reflects the realization that a bad topology may result in poor performance. A group of researchers, led by Mendes [25, 31, 32, 33], have studied the effects of using different population topologies in the performance of the canonical PSO as well as in the fully informed particle swarm. Their results show that there are interactions between the population topology and the problem being solved. For some kind of problems, a fully connected topology works best, for others, a ring topology does. In a recent study in which dynamically changing topologies as well as randomly generated topologies were tested, Mohais et al. [34] conclude that random topologies have the same or even better performance than nonrandom topologies.

2.4.3 Hybridizations

As any other stochastic optimization technique, PSO has some strengths and weaknesses. An active research area in the field of optimization in general, is the hybridization of algorithms. The underlying hypothesis is that two or more algorithms' strengths can be combined to produce a single more powerful algorithm.

Many works in this research area have dealt with the integration of some operators from evolutionary algorithms into PSO. For example, Angeline [2] proposed the use of selection in PSO to improve its effectiveness. In his proposal, the best half of the particles population was selected and copied into the worst half. Different mutation operators have also been tried in PSO [15, 47, 53] to preserve some diversity among particles. A reproduction scheme in PSO was introduced by Løvbjerg et al. [27]. They use the arithmetic crossover operator which is commonly used in real-valued genetic algorithms. Additionally they divide the population into subpopulations in order to preserve some diversity.

Another approach towards the hybridization of algorithms is to use one algorithm as a component of another one. One example of such approach is the work of Juang [20] in which a genetic algorithm uses a PSO algorithm to enhance the elite of the population. The inspiring metaphor is that individuals can become fitter (from an evolutionary perspective) during their lifetime and that not only genetic material inherited from their parents determine their chance of survival.

Finally, we would like to mention that PSO has also been enriched with concepts derived from the other prominent exponent of swarm intelligent systems; namely, Ant Colony Optimization (ACO) [9]. Holden and Freitas [16] combined the power of PSO for optimizing real-valued functions with the power of ACO for optimizing discrete variables. The resulting algorithm was capable of dealing with discrete and continuous variables at the same time. They applied this algorithm to the discovery of classification rules for hierarchical data.

Chapter 3

An Empirical Evaluation Methodology of Stochastic Optimization Algorithms

In this chapter, we describe the empirical evaluation methodology we used to assess the performance of the different Particle Swarm Optimization (PSO) algorithms we included in our study. This empirical methodology is the one proposed by Hoos and Stützle [17] and is based on what they call *run-time and solution-quality distributions*. We describe this methodology along with its motivation and advantages. We first explain the foundations of the method and give some examples of its application. After that, we compare the adopted methodology with the one that is currently in use by the PSO community.

3.1 Stochastic Optimization Algorithms and their Run-Time and Solution-Quality Distributions

Most stochastic optimization algorithms try to find an optimal solution to a problem in an iterative way. Therefore, and since the exploration of the search space is heuristic, we normally do not know when to stop a run of the optimization algorithm. The problem is that we might either, stop the run prematurely without getting any useful result, or that we may be wasting computing time by making the algorithm run for longer when it has already reached a sufficiently good solution.

However, there may be situations in which we have some knowledge about the problem. Normally, we can use this knowledge to establish sensible termination criteria. For example, we might know the objective function value of the optimal solution or we might have an estimation of it. In this case, we can stop a run of an optimization algorithm whenever the objective function value of the solution found by the algorithm is close enough to the goal.

There may be other cases in which the evaluation of the objective function is so costly, that we cannot afford many function evaluations in our search for the optimal solution. In this case, a maximum number of objective function evaluations may be the termination criterion. We can generalize this case to situations in which we cannot spare resources (including time) in the optimization process.

For convenience, let us label this cases as scenarios under which we may apply a stochastic optimization algorithm:

Scenario 1. We have some knowledge (e.g., in the form of an estimation of the objective function value of the optimal solution) about the problem and our main concern is to find, as fast as possible, a solution whose quality is close enough to the goal.

Scenario 2. We have scarce resources and we want the best possible solution given a fixed time or computational effort limit.

In case of scenario 1, the time required by an implementation of a stochastic optimization algorithm to find a solution of a certain quality can be modeled as a random variable. Indeed, if we run a stochastic optimization algorithm twice or more times using as stopping criterion a minimum acceptable solution quality, we will see that the time we have to wait for the algorithm to return a solution is different from run to run. Similarly, in cases type 2, if we run the same algorithm several times using a maximum number of objective function evaluations as stopping criterion, the quality of the returned solution will vary from run to run. The quality of the solution returned by the algorithm given a fixed time or computational effort limit can also be modeled as a random variable.

The probability distributions of the run-time and solution-quality of a stochastic optimization algorithm applied to a particular problem can give useful information about the performance of the algorithm. Additionally, they can help in the identification of weaknesses of the algorithm and thus, they can be used as analysis tools of stochastic optimization algorithms. Since these probability distributions cannot be analytically derived, we discuss the issue of empirically estimating them in the following sections.

3.1.1 Estimating Run-Time Distributions

Let T_q be the random variable describing the time needed by a stochastic optimization algorithm to find a solution of quality q. The cumulative distribution function $RT_q(t)$ of T_q is called the *run-time distribution* and it is defined as

$$RT_q(t) = P(T_q \leq t)\,, \tag{3.1}$$

where the right-hand side represents the probability that T_q takes a value less than or equal to t.

There is an alternative way of measuring computational effort besides using computing time, and that is counting the number of times the most expensive operations are performed. In most practical applications, the evaluation of the objective function is the most expensive operation and therefore, it is common practice to use this number as a measure of computational effort. When using the number of critical operations instead of time, we are interested in what is called a *run-length distribution*.

Formally, a run-length distribution is defined as

$$RL_q(l) = P(L_q \leq l)\,, \tag{3.2}$$

where L_q is the random variable describing the number of critical operations needed by a stochastic optimization algorithm to find a solution of quality q. $P(L_q \leq l)$ is the probability that L_q takes a value less than or equal to a certain number of critical operations l.

In order to estimate the run-time or run-length distribution of an algorithm on a particular problem, it suffices to run the algorithm several times and record information relative to solution improvement. In every run, we need to record the time or the number of critical operations, and the solution quality whenever a new best solution is found by the algorithm.

Assuming that the data have been collected by running n times (using a different random number generator seed in every run) the same algorithm on the same problem, the empirical run-time distribution can be computed as follows

$$\widehat{RT_q(t)} = \frac{1}{n} \sum_{i=1}^{n} I(rt_i(q) \leq t)\,, \tag{3.3}$$

where I is an indicator function defined as

$$I(x \leq y) = \begin{cases} 1 & \text{if, indeed, } x \leq y \\ 0 & \text{otherwise} \end{cases}\,, \tag{3.4}$$

and $rt_i(q)$ is the time required by the i-th run to find a solution of quality at least as good as q.

The empirical run-length distribution is estimated in a very similar way as the run-time distribution. It requires to substitute $rt_i(q)$ for $rl_i(q)$ in Equation 3.3. The function $rl_i(q)$ is defined as the number of critical operations needed by the i-th run to find a solution of quality q or better.

The solution quality threshold q can be specified as an absolute value or as a relative one. Relative solution quality thresholds can be specified as percentage deviations from the known or estimated optimum value. In this case, $q\,[\%] = 100 \cdot (f(s) - f(s_{opt}))/f(s_{opt})$, where $f(s)$ is the value of a certain solution s, and $f(s_{opt})$ is the value of the optimum. In order to avoid a division by zero, the objective function can be biased by some constant factor c, so that $f(s) = g(s) + c$, where g is the original objective function.

It is impossible to estimate the run-time distribution over the full domain of RT_q, that is, $\mathbb{R}^+$. This is obvious since we cannot run an algorithm for eternity. In practice, we have to set a maximum time limit, or in the case of run-length distributions, a maximum number of critical operations. However, for most practical cases, a sufficiently large limit will suffice to have an idea of the performance behavior of the algorithm under study.

Figure 3.1 shows an example of empirical run-length distributions of a stochastic optimization algorithm[1]. It shows run-length distributions at different solution qualities. These empirical distributions were computed using 100 independent trials each running for 1 000 000 function evaluations.

In run-time or run-length plots, the x-axis represents time or the number of critical operations respectively, and the y-axis represents the probability of solving the problem at the desired quality level. We defer the topic of interpreting run-time or run-length distribution plots to Section 3.1.3.

[1]Details about the problem and the algorithm are unimportant for the presentation purpose.

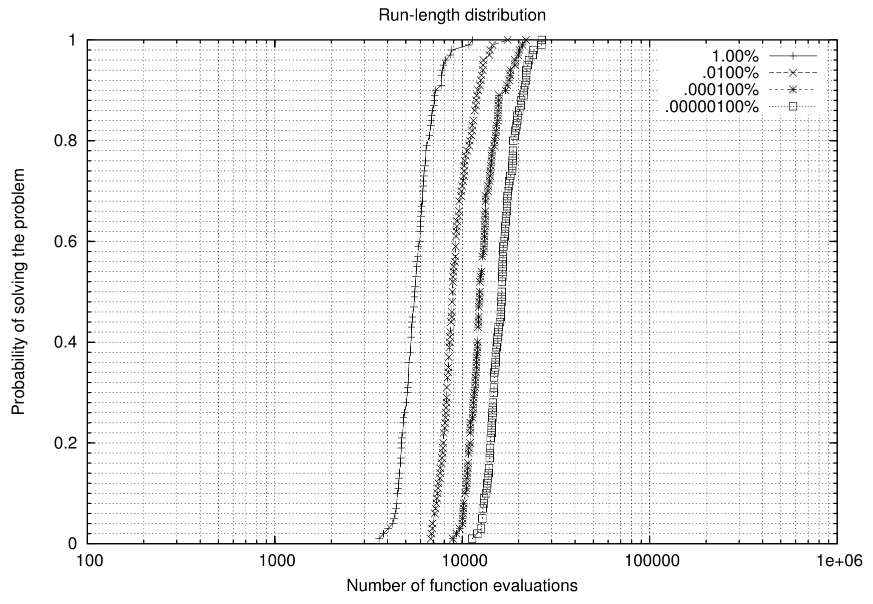


Figure 3.1: Example of an empirical run-length distribution. The plots correspond to run-length distributions at different solution qualities.

3.1.2 Estimating Solution-Quality Distributions

The solution-quality distribution of a stochastic optimization algorithm applied to a particular problem is defined as

$$SQ_t(q) = P(Q_t \leq q)\,, \tag{3.5}$$

where Q_t is a random variable that represents the solution quality achieved by an algorithm exactly after some effort limit t (time or number of critical operations). $P(Q_t \leq q)$ is the probability that Q_t takes on a value less than or equal to q. If q is specified as the percentage deviation from a goal, the smaller q, the better. From the definition, it is clear that SQ_t is the cumulative probability distribution of Q_t.

In order to estimate the solution-quality distribution, we need exactly the same data for estimating the run-time distribution. The expression we need to compute the empirical solution-quality distribution is also very similar:

$$\widehat{SQ_t(q)} = \frac{1}{n}\sum_{i=1}^{n} I(sq_i(t) \leq q)\,, \tag{3.6}$$

where n is the number of independent trials, I is an indicator function as defined in Equation 3.4, $sq_i(t)$ is the solution quality achieved by the i-th run at time (or number of function evaluations) t.

Figure 3.2 shows examples of solution-quality distributions at different computational effort limits. These solution-quality distributions correspond to a 20-particle canonical particle swarm optimizer using a ring population topology applied to a 30-dimensional Rastrigin problem.

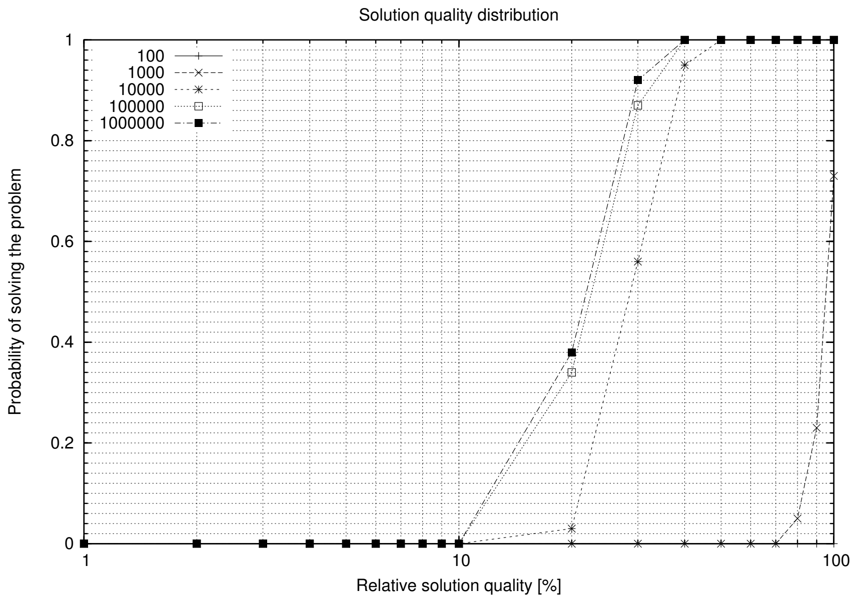


Figure 3.2: Example of an empirical estimation of solution-quality distributions. The plots correspond to the solution-quality distributions at different effort limits (measured as objective function evaluations) of a 20-particle canonical particle swarm optimizer using a ring population topology applied to a 30-dimensional Rastrigin problem.

In solution-quality distribution plots, the x-axis corresponds to the solution quality (either absolute or relative), and the y-axis corresponds to the probability of finding a solution of a required quality given some fixed effort limit.

3.1.3 Interpretation of Run-Time Distributions

Let us suppose for a moment that we are interested in the run-time behavior of a deterministic optimization algorithm applied to some problem. From the very nature of the algorithm, we would observe that the time or effort needed to find a solution of a particular quality is always the same no matter how many times we run the algorithm[2]. In a run-time or run-length distribution graph, we would see a vertical line directly above the time or effort required to find the solution of the desired quality. Of course, this will be the case if the deterministic algorithm is capable of finding a solution of the desired quality in first place.

The case described above is an extreme case in the shape of run-time distributions. Another extreme case is when an algorithm is incapable of finding at least one solution of the desired quality within the time or effort limits allocated for the empirical estimation. In this case, we can draw either of the following conclusions: (i) the algorithm is simply not capable of finding a solution of the desired quality within the allocated estimation range, or (ii) a solution of the desired quality does not exist. However, as we will see, there is sometimes a "trend" in the shape and position of run-time or run-length distributions that

[2]Neglecting random effects due to other factors, such as CPU load.

can help us guess how a distribution would look like for another solution quality threshold. In any case, since the necessary data to estimate a run-time or run-length distribution is the same for any solution quality, we can always confirm our guesses.

There is a third case in which the estimated distribution seems to be incomplete. This sometimes happens when some runs simply "got lost" and did not find a solution of the desired quality within the allocated time or effort limit. This phenomenon is of particular interest when we want to improve the performance of an algorithm. We will discuss more about this later in this section.

In general, we can expect run-time or run-length distributions to have shapes that go from almost vertical to almost horizontal both in cut or uncut forms. Figure 3.3 shows some examples of empirical run-length distributions of different shapes.

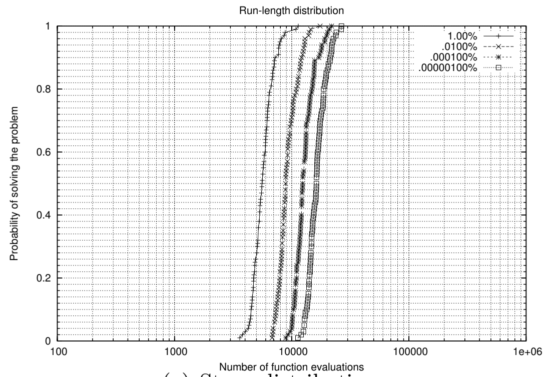
Run-time distributions provide information about the behavior of a stochastic optimization algorithm in two main respects at least. First, they can tell us how easy/hard is to find the desired solution quality in the specific problem the algorithm is applied to, and second, they can tell us how prone the algorithm is to stagnating. Run-time distributions are therefore, useful for comparing the performance of stochastic optimization algorithms.

If the probability of finding a desired solution quality is one after just a few objective function evaluations, we can say that the requirement was easy for the algorithm to fulfill. On the other hand, if the probability is one but after many more evaluations, we can say that the requirement is difficult to fulfill. Figure 3.4 shows examples of run-length distributions for easy, moderate and difficult solution quality requirements. In general, the easier the requirement, the more to the left the distribution will tend to appear. Likewise, the harder the requirement, the more to the right a distribution will appear.

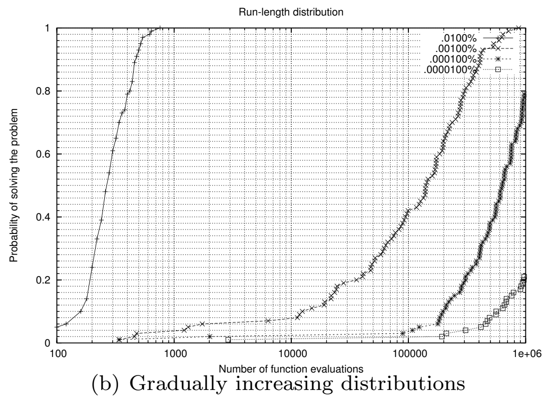
Note that in order to be considered easy, moderately difficult or hard, the bulk of the distribution needs to be concentrated in the same region (e.g., Figure 3.3(a)). If a distribution covered uniformly the x-axis, we could not say anything about the easiness/hardness of the solution quality requirement.

With respect to the propensity of an algorithm to stagnate on a particular problem, run-time or run-length distributions help us to qualitatively determine the severity of the stagnation. An algorithm with propensity to stagnate will show a slowly-increasing or a non-increasing probability (towards the right tail of the distribution) of finding a solution of a given quality over the allocated time limit. Cut distributions such as the ones shown in Figure 3.3(c) are examples of algorithms with severe stagnating behavior. These algorithms sometimes find a solution in a reasonable time, but there are also occasions in which, even after letting them run for much longer, no solution is found. A possible explanation to this phenomenon is that the algorithm gets trapped in a region of the search space that contains no solutions (with the desired quality level).

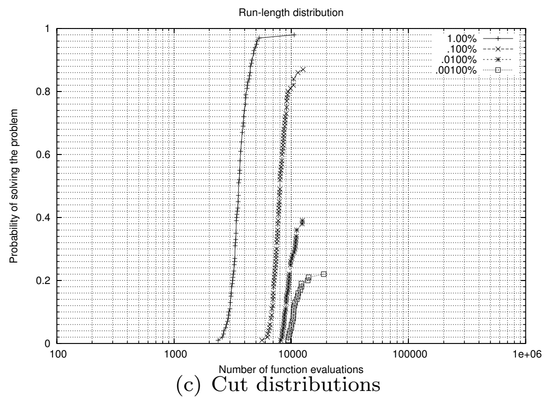
An example of a slowly-increasing run-time distribution is shown in Figure 3.5. In this graph, we can see how in three out of four desired solution qualities, the algorithm that produced the data for these distributions suffers from a strong stagnating behavior. Note how the probability increases but at an extremely slow rate (recall that the x-axis is in logarithmic scale). The probability of finding a solution of 0.001% away from the optimum is about 0.26 after 10 000 function evaluations, and only 0.34 after 100 000 evaluations.



(a) Steep distributions



(b) Gradually increasing distributions



(c) Cut distributions

Figure 3.3: Empirical run-length distributions of different shapes. Figure 3.3(a)shows a steep distribution across different solution qualities. Figure 3.3(b) shows a gradually increasing distribution across solution qualities. Figure 3.3(c) shows cut distributions across solution qualities.

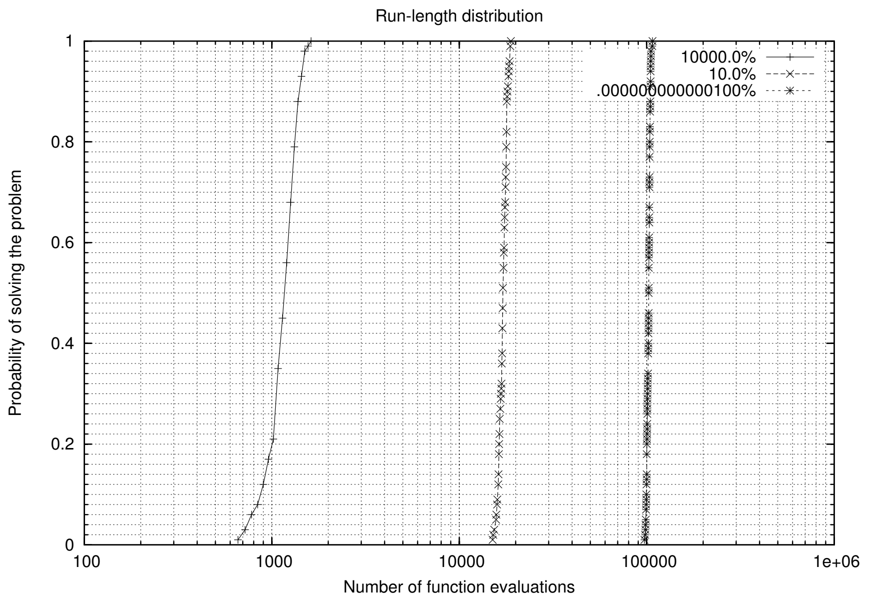


Figure 3.4: Run-length distributions showing easy, moderate and difficult solution quality requirements. The leftmost distribution corresponds to the easy case, the one in the center corresponds to the moderately difficult case, and the rightmost distribution corresponds to the hardest requirement.

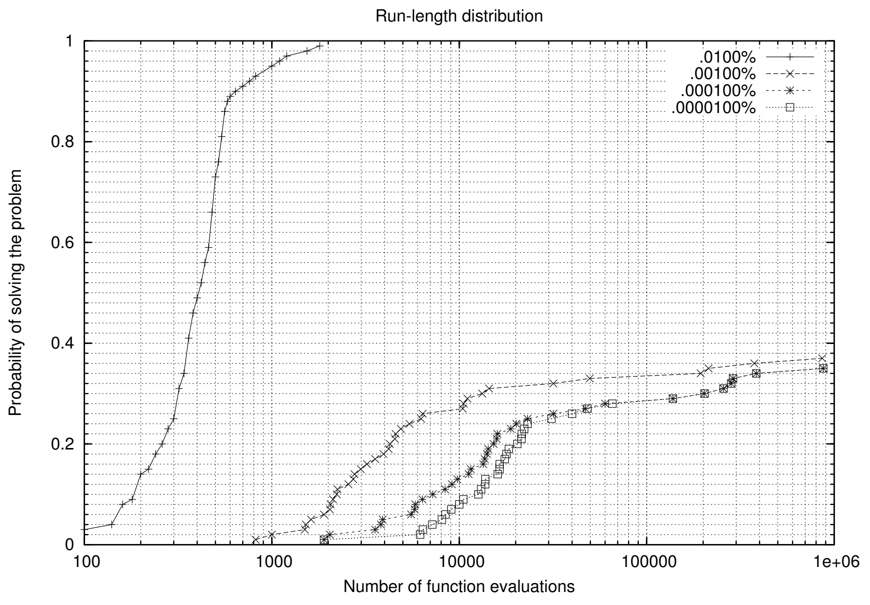


Figure 3.5: Example of a run-length distribution showing an algorithm with strong stagnating behavior. Three out of four solution qualities show how the algorithm that produced these distributions exhibits strong stagnating behavior. Note the logarithmic scale in the x-axis.

We said before that run-time or run-length distributions provide us information that could be used to improve the performance of an algorithm. If after estimating the run-time or run-length distribution of an algorithm we see that it tends to stagnate, we could either complement it with an extra diversification mechanism or we could promote its intensification behavior. How the analysis of run-time distributions can suggest improvement strategies is presented in more detail in Chapter 5.

3.1.4 Interpretation of Solution-Quality Distributions

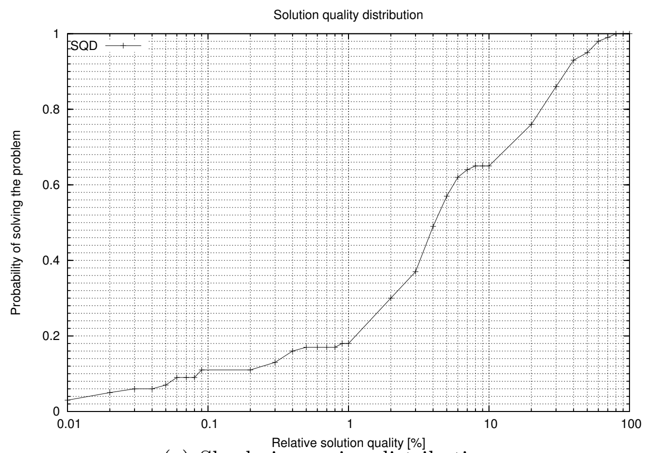
If we run a deterministic optimization algorithm twice or more times with a maximum effort limit, it will always return a solution with a certain quality level. Therefore, the solution-quality distribution of a deterministic optimization algorithm will resemble a step function that will have a zero value for solution qualities that are better than the best solution quality achievable by the algorithm given a maximum computational effort limit[3].

A stochastic optimization algorithm is expected to exhibit solution-quality distributions with increasing probability towards quality levels away from the optimum. A slowly increasing probability is evidence of a high variability in the solution quality achievable by the algorithm given a fixed effort limit. It is also expected that if the algorithm is given more resources (i.e., more time or more objective function evaluations) to use, the probability of finding high quality solutions increases. Here, the phenomenon of stagnation can be detected if the solution-quality distributions (for two very different computational effort limits) superimpose or if they are very near from each other. Figure 3.6 shows one example of each of these two cases.

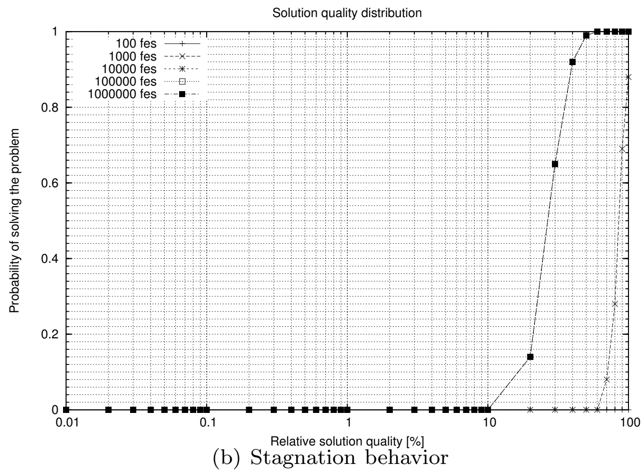
In general, if we need high quality solutions, we must be prepared to allocate more resources to the optimization algorithm. Of course, as we have seen, if the optimization algorithm exhibits a strong stagnating behavior, resources may end up being wasted.

Solution-quality distributions can also reveal an interesting phenomenon. An algorithm can get trapped in a suboptimal region of the search space from time to time and its solution-quality distributions can tell us the quality level of this region and with what probability the algorithm is capable of escaping from it. Figure 3.7 shows a series of solution-quality distributions in which it is clear the quality level of the "trap" in the search space and the probability of escaping from it for different effort limits. In this example, the "trap" lies somewhere in between 0.003 and 0.004% away from the known optimum. The escape probabilities are 0.56, 0.54, 0.36, and 0.08 for 1 000 000, 100 000, 10 000, and 1 000 maximum objective function evaluations respectively. For 100 maximum objective function evaluations the algorithm is not even able to reach the "trap" region and therefore there is no escape probability associated with it.

[3]Recall, that if solution qualities are specified as percentage deviations, lower percentages represent better solutions.



(a) Slowly increasing distribution



(b) Stagnation behavior

Figure 3.6: High variability of solution quality levels and stagnation in solution-quality distributions. Figure 3.6(a) shows a slowly increasing solution-quality distribution. Sign of a high variability in achievable solution quality levels. Figure 3.6(b) shows how the distributions for 10 000 function evaluations or more, superimpose each other. A sign of stagnating behavior. The label "fes" refers to the number of function evaluations

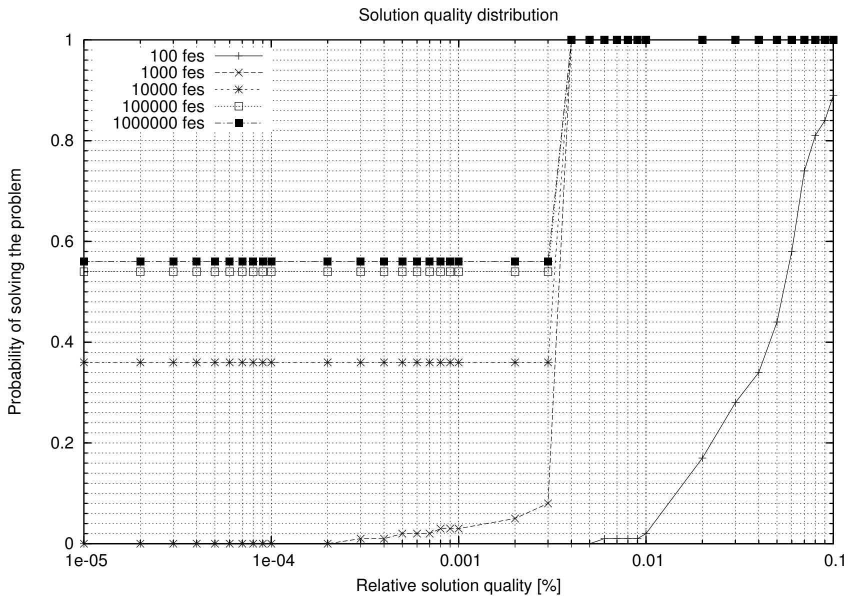


Figure 3.7: A solution-quality distribution revealing a "trap"' in the search space and the escape probability for different effort limits. The "trap" lies somewhere between 0.003 and 0.004% away from the known optimum. The escape probabilities are 0.56, 0.54, 0.36, and 0.08 for 1 000 000, 100 000, 10 000, and 1 000 maximum objective function evaluations respectively.

3.2 Run-Time and Solution-Quality Distributions and Current Evaluation Practices

Within the PSO community, as well as within the optimization community in general, people are interested in optimization algorithms that return high-quality solutions with the least possible computational effort. Having this goal in mind, researchers evaluate the performance of stochastic optimization algorithms by asking the following three questions:

- How many critical operations or iterations does the algorithm need to achieve a certain solution quality level?
- How often is the algorithm capable of achieving the required solution quality goal?
- What is the solution quality the algorithm is capable of achieving given a fixed effort limit?

The first and second questions are related to application scenarios type 1 (see Section 3.1), and the third question is related to application scenarios type 2. In this respect, current practices do not differ from the methodology adopted for our study. Both methodologies have the same concerns; however, they differ in the way they try to give answer the above posted questions.

To evaluate an algorithm considering application scenarios type 1, current evaluation practices fix a goal in the solution quality level of a problem that the algorithm is supposed to reach. Then the algorithm is run a number of times and the average number of iterations or function evaluations needed to reach the goal is reported. Some measure of dispersion, such as the standard deviation, is also commonly reported. Based on the results, statements such as “parameterization A yields a faster algorithm than parameterization B”, or “algorithm A is faster than algorithm B” are not rare.

There is a high risk of drawing wrong conclusions from this kind of evaluation. A fast convergence rate for goal A does not necessarily implies a fast convergence rate for goal B. To alleviate this problem, researchers generate *solution-quality development over time* (SQT) plots. These plots show the average solution quality achieved at some iteration or function evaluation number. These plots provide much more information than fixed-time statistics and allow a fairer comparison. There is a problem with average-based SQT plots. We have seen before that some runs cannot get to some (usually high-quality) solutions. A possible solution is to average over successful runs (runs that indeed reached the goal) only, but this measure can be misleading. Another problem is that the average is very sensitive to extreme values.

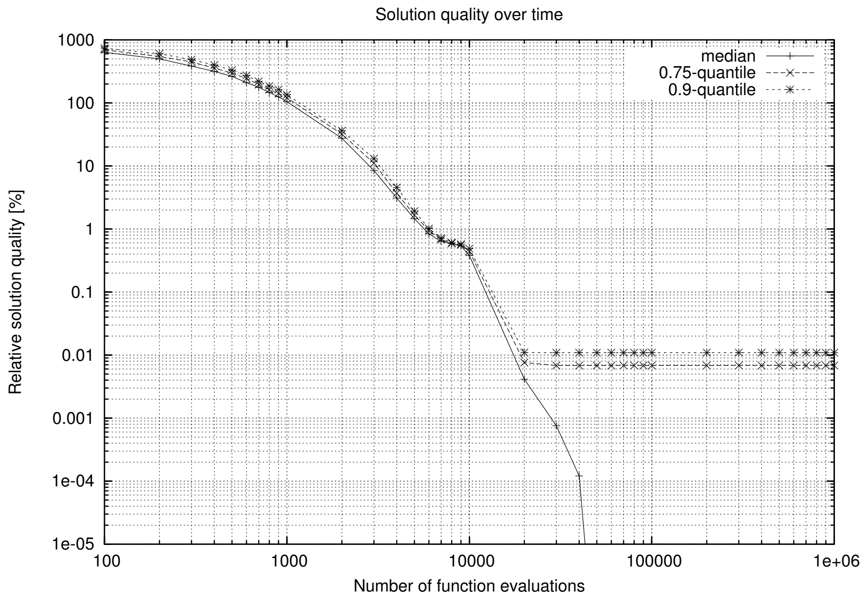


Figure 3.8: Example of solution quality development over time plots based on run-time and solution-quality distributions. It shows the 0.5- (median), 0.75- and 0.9-quantiles of the bi-variate distribution of (T_q, Q_t).

The solution based on run-time and solution-quality distributions is to generate SQT plots based on certain quantiles of the distribution of the composite (T_q, Q_t) random variable. Recall that T_q is a random variable describing the time needed by a stochastic optimization algorithm to find a solution of quality q, and Q_t is a random variable that represents the solution quality achieved by an algorithm exactly after some effort limit t. These plots are insensitive to extreme values and implicitly deal with unsuccessful runs. Figure 3.8 shows an example SQT plot based on the 0.5- (median), 0.75- and 0.9-quantiles of the bi-variate distribution of (T_q, Q_t).

From the analysis of the plots, it can be seen that 90% of the time, the algorithm will achieve a solution that is, in terms of quality, closer than 0.01% away from the optimum in runs that use up to 1 000 000 function evaluations. This information would be impossible to obtain using only solution quality statistics at fixed points in time or average-based SQTs.

The second question in the list is usually answered by counting the number of times the algorithm succeeded in reaching the predefined goal and dividing this number by the total number of trials. The result is usually referred to as the *success rate*. Another possible interpretation of run-time or run-length distributions is as the *development of the success rate over time*. Indeed, estimating a run-time or run-length distribution is equivalent to computing the success rate of achieving a certain solution quality level, but at different effort limits.

In current evaluation practices the third question is addressed in a similar way as the first question. The algorithm is run for a fixed number of iterations or function evaluations and the achieved solution quality is averaged. This measure is very often used to draw conclusions in algorithms comparisons. Statements such as "algorithm A outperformed algorithm B because A got an average solution quality of 0.001 while B only got 0.1" are very tempting when looking at this kind of statistics. Using solution-quality distributions we are able see the distribution and therefore we can be more careful when making this kind of statements.

Chapter 4

Empirical Evaluation of Particle Swarm Optimizers

In this chapter, an empirical evaluation of the particle swarm optimizers described in Section 2.3 is presented. We begin the presentation by describing the experimental setup we used in our evaluation. Then, a comparison among the evaluated algorithms is presented. The last section is devoted to the analysis of different inertia weight schedules in the particle swarm optimizer with time-decreasing inertia weight.

4.1 Experimental Setup

Our experimental setup was designed with one objective in mind. We wanted to test several particle swarm optimizers under the assumption that no *a priori* knowledge about the structure of the problem was available. This is the situation that we face when we are given a real-world problem.

In general, our first approach would be to use the state-of-the-art algorithm with what are considered "normally good" parameters. For this reason, in our experimental setup, each algorithm used the same parameterization across all benchmark problems.

The set of benchmark problems and the actual algorithms' parameter values that we used in our experiments are described below.

4.1.1 Benchmark Problems

The benchmark functions that we used in the study presented in this document are among the most widely used for assessing the performance of evolutionary algorithms. Even though some of them are considered "easy" problems, they help us understand how algorithms explore the search space. It is clear that the goal is not to solve benchmark problems to optimality, but rather to understand how different particle swarm optimizers behave under different circumstances.

All the benchmark functions that we used in our evaluation have their global optimum displaced and biased. In most of the cases, we used exactly the same values that were proposed in the set of benchmark functions used for the special session on real parameter optimization of the IEEE Congress of Evolutionary

Computation 2005 [55]. The exact displacement values can be found in Appendix A.

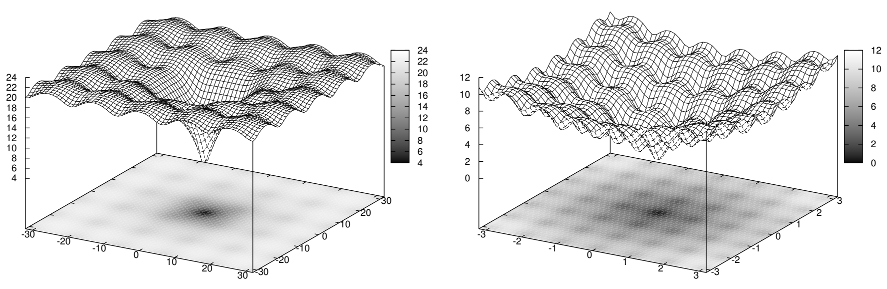
In our experiments, we used 30-dimensional functions except in the case of Easom and Schaffer functions that are two-dimensional by definition. In the following, we describe in some detail each of the benchmark functions used in our study.

Ackley Function

This function is named after Ackley [1] who invented it. The original version was a two-dimensional function and it was later generalized to n dimensions by Bäck [3]. The general form is defined as

$$f(\vec{x}) = -20e^{-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}} - e^{\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)} + 20 + e\,, \tag{4.1}$$

where $\vec{x}$ is an n-dimensional vector that is normally located within the range $[-32.0, 32.0]^n$. The global minimum is located at the origin and its value is zero. Figure 4.1 shows the Ackley function in two dimensions. On the left, that is, in Figure 4.1(a) we can see the shape of the function over the whole search range. In Figure 4.1(b) we can observe a zoom near the global optimum.



(a) Ackley (whole range) (b) Ackley (zoom)

Figure 4.1: Ackley Function. In (a) a two-dimensional Ackley function over the whole search range. In (b) a zoom of the region that contains the global optimum.

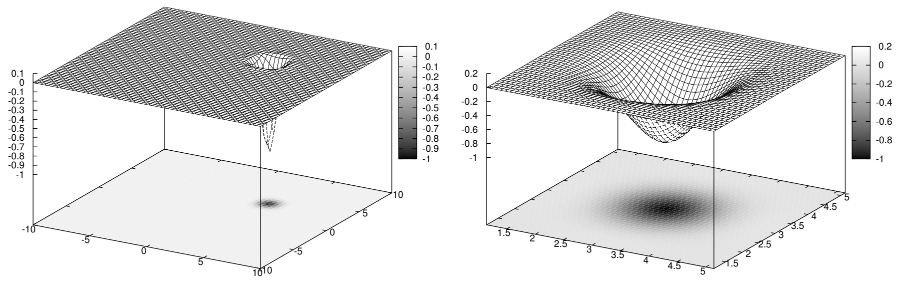
The Ackley function is a nonlinear multimodal function with regularly distributed local optima.

Easom Function

This function was proposed by Easom [10] to evaluate global optimization techniques. It is a n-dimensional function with a single minimum that is also the global optimum. The mathematical expression that defines it is

$$f(\vec{x}) = -\prod_{i=1}^{n}\cos(x_i)\cdot e^{-\sum_{i=1}^{n}(x_i-\pi)^2}\,, \tag{4.2}$$

where $\vec{x}$ is an n-dimensional vector normally located within the range $[-10.0, 10.0]^n$. The global minimum is located at the point $(\pi, \ldots, \pi)$ and its value is -1. Figure 4.2 shows a two-dimensional Easom function.



(a) Easom (whole range) (b) Easom (zoom)

Figure 4.2: Easom Function. In (a) a two-dimensional Easom function over the whole search range. In (b) a zoom of the region that contains the global optimum.

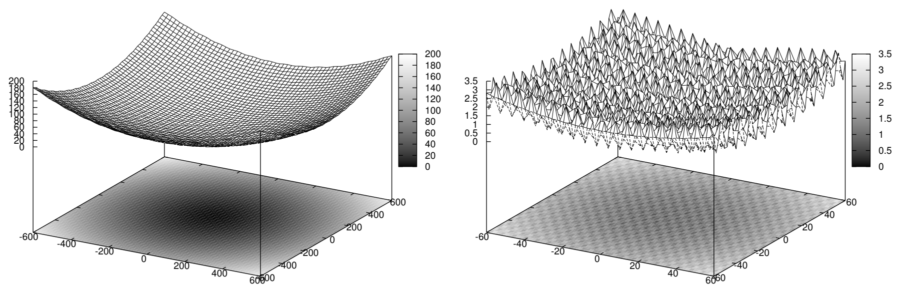
The Easom function is characterized by a large plateau compared to the size of the region that contains the global minimum. It is usually used in its two-dimensional form.

Griewank Function

The mathematical formula that defines this function is

$$f(\vec{x}) = \frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n}\cos\left(\frac{x_i}{\sqrt{i}}\right) + 1\,, \tag{4.3}$$

where $\vec{x}$ is an n-dimensional vector located within the range $[-600.0, 600.0]^n$. The global minimum is located at the origin and its value is zero. Figure 4.3 shows a two-dimensional Griewank function.



(a) Griewank (whole range) (b) Griewank (zoom)

Figure 4.3: Griewank Function. In (a) a two-dimensional Griewank function over the whole search range. In (b) a zoom of the region that contains the global optimum.

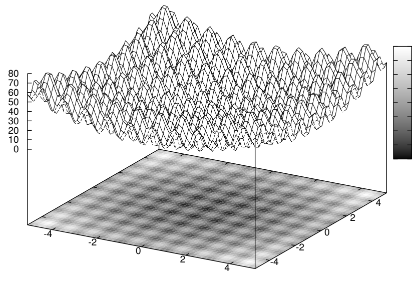
This function has many regularly distributed local minima. However, as n increases the number of local minima decreases [57].

Rastrigin Function

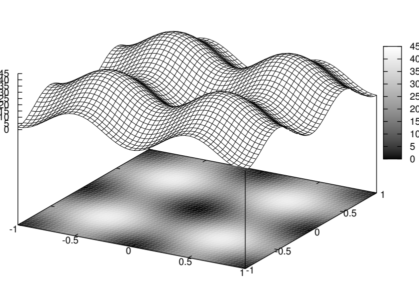
Named after Rastrigin [41], this function is mathematically defined as

$$f(\vec{x}) = 10n + \sum_{i=1}^{n} \left(x_i^2 - 10\cos(2\pi x_i) \right) \ , \tag{4.4}$$

where where $\vec{x}$ is an n-dimensional vector located within the range $[-5.12, 5.12]^n$. The global minimum is located at the origin and its function value is zero. Figure 4.4 shows a two-dimensional Rastrigin function.



(a) Rastrigin (whole range)



(b) Rastrigin (zoom)

Figure 4.4: Rastrigin Function. In (a) a two-dimensional Rastrigin function over the whole search range. In (b) a zoom of the region that contains the global optimum.

This function is based on the Sphere function (see Section 4.1.1 below) with a modulator term $\alpha\cos(2\pi x_i)$ which induces a "wave" over its surface. The location of local minima is regularly distributed.

Rosenbrock Function

This function was invented by Rosenbrock [45], and is mathematically defined as

$$f(\vec{x}) = \sum_{i=1}^{n-1} \left(100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2 \right) \ , \tag{4.5}$$

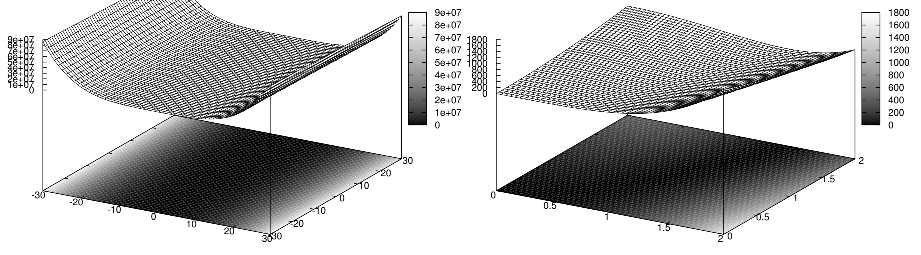
where $\vec{x}$ is an n-dimensional vector located within the range $[-30.0, 30.0]^n$. The global optimum is located at $(1, \ldots, 1)$ with a function value of zero. Figure 4.5 shows a two-dimensional Rosenbrock function.

This function exhibits a parabolic-shaped deep valley. In the optimization literature it is considered a difficult problem due to the nonlinear interaction between variables [35].

Salomon Function

This function is rotation-invariant and was proposed by Salomon [46]. It is defined as

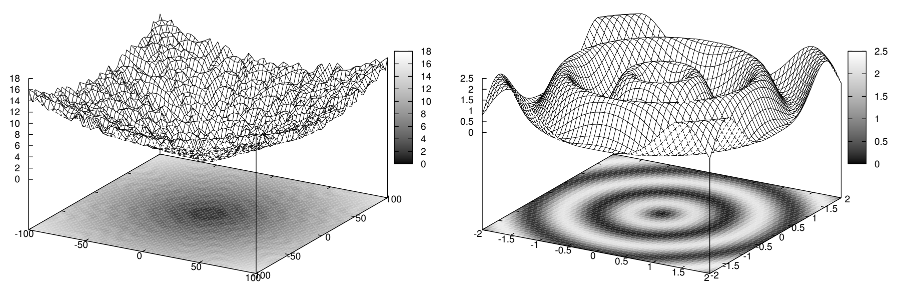
$$f(\vec{x}) = 1 - \cos\left(2\pi\sqrt{\sum_{i=1}^{n} x_i^2}\right) + 0.1\sqrt{\sum_{i=1}^{n} x_i^2} \ , \tag{4.6}$$



(a) Rosenbrock (whole range) (b) Rosenbrock (zoom)

Figure 4.5: Rosenbrock Function. In (a) a two-dimensional Rosenbrock function over the whole search range. In (b) a zoom of the region that contains the global optimum.

where $\vec{x}$ is an n-dimensional vector located within the range $[-100.0, 100.0]^n$. The global optimum is located at the origin with a function value of zero. Figure 4.6 shows a two-dimensional Salomon function.



(a) Salomon (whole range) (b) Salomon (zoom)

Figure 4.6: Salomon Function. In (a) a two-dimensional Salomon function over the whole search range. In (b) a zoom of the region that contains the global optimum.

Schaffer Function

This function is also known as Schaffer's F6 or as the sine envelope sine wave. It is mathematically defined as

$$f(\vec{x}) = 0.5 + \frac{\sin^2\left(\sqrt{x_1^2 + x_2^2}\right) - 0.5}{\left(1 + 0.001(x_1^2 + x_2^2)\right)^2}\,, \tag{4.7}$$

where $\vec{x}$ is an two-dimensional vector located within the range $[-100.0, 100.0]^n$. The global optimum is located at the origin with a function value of zero. Figure 4.7 shows the Schaffer function.

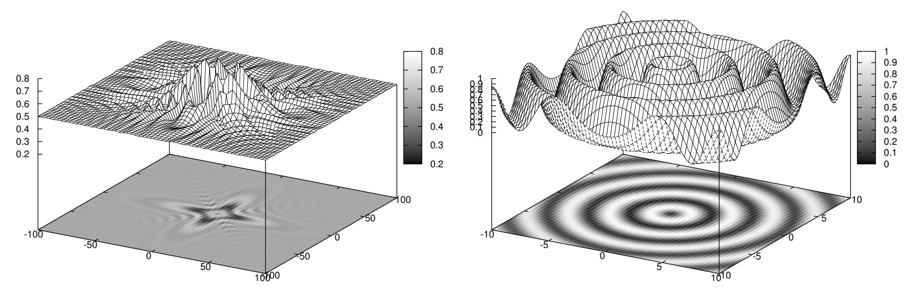

(a) Schaffer (whole range) (b) Schaffer (zoom)

Figure 4.7: Schaffer Function. In (a) a two-dimensional Schaffer function over the whole search range. In (b) a zoom of the region that contains the global optimum.

Even though it is a two-dimensional problem, the global optimum of this function is usually hard to identify because the local gradients do not provide much information about the location of the global optimum.

Schwefel Function

This function is also known as Schwefel's sine root function. It is defined as

$$f(\vec{x}) = \sum_{i=1}^{n} -x_i \sin\left(\sqrt{|x_i|}\right) \ , \tag{4.8}$$

where $\vec{x}$ is an n-dimensional vector located within the range $[-512.0, 512.0]^n$. The global optimum is located at $(420.9687, \ldots, 420.9687)$ with a function value of zero. Figure 4.8 shows a two-dimensional Schwefel function.

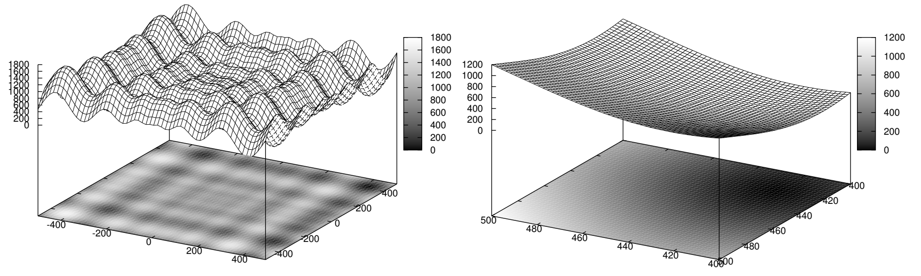

(a) Schwefel (whole range) (b) Schwefel (zoom)

Figure 4.8: Schwefel Function. In (a) a two-dimensional Schwefel function over the whole search range. In (b) a zoom of the region that contains the global optimum.

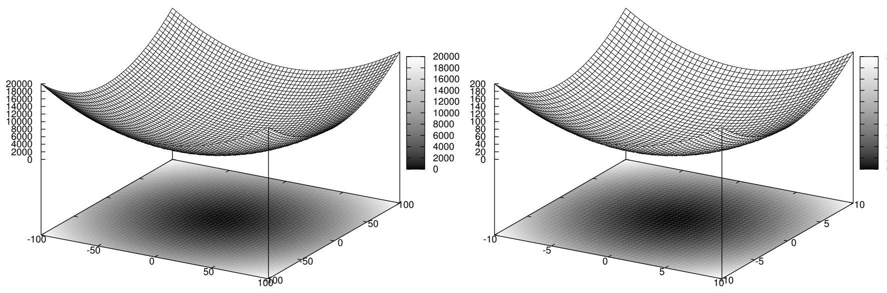
The main difficulty of this function is that the second best minimum is very far from the global optimum. In this function, contrary to the case of all the previous functions, the search range is very important. Outside the search range, the Schwefel function can take on values lower than the optimum of interest within the search range.

Sphere Function

This function is defined as

$$f(\vec{x}) = \sum_{i=1}^{n} x_i^2 \,, \tag{4.9}$$

where $\vec{x}$ is an n-dimensional vector located within the range $[-100.0, 100.0]^n$. The global optimum is located at the origin with a function value of zero. Figure 4.9 shows a two-dimensional Sphere function. The Sphere function is a



(a) Sphere (whole range) (b) Sphere (zoom)

Figure 4.9: Sphere Function. In (a) a two-dimensional Sphere function over the whole search range. In (b) a zoom of the region that contains the global optimum.

highly convex, unimodal test function. Not performing well in this function is considered bad since any gradient descent method would solve this problem effectively.

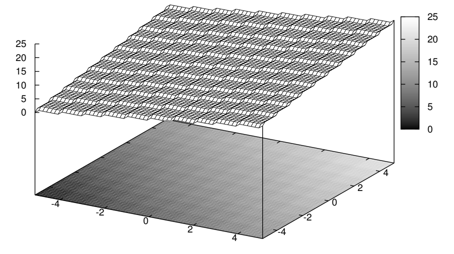
Step Function

This function is mathematically defined as

$$f(\vec{x}) = 6n + \sum_{i=1}^{n} \lfloor x_i \rfloor \,, \tag{4.10}$$

where $\vec{x}$ is an n-dimensional vector located within the range $[-5.12, 5.12]^n$. The global optimum is located at $([-5.12, 5.0), \ldots, [-5.12, 5.0))$ with a function value of zero. Figure 4.10 shows a two-dimensional Step function.

The step function is a piece-wise continuous step function with many flat surfaces. Flat surfaces are considered a problem since they do not provide any information about the location of the global optimum. The size of the region where the global optimum is located is small compared to the size of the entire search range. This can be seen in Figure 4.10(b).


(a) Step (whole range)

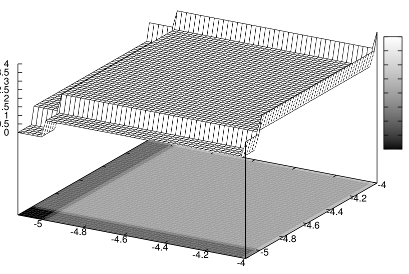

(b) Step (zoom)

Figure 4.10: Step Function. In (a) a two-dimensional Step function over the whole search range. In (b) a zoom of the region that contains the global optimum.

4.1.2 Parameter Settings

In accordance to our goals, we decided to test the particle swarm optimizers that we included in our study without tuning their set of parameters specifically for each test problem. Instead, all algorithms were run with the same set of parameters over all test problems. The specific parameter settings were those that are normally used in the literature.

Since in our study we vary some parameters to measure their effects on the algorithms' performance, the remaining parameters remained fixed. The choice was based on existing literature. Table 4.1 lists the algorithms' fixed parameter settings that we used in our experiments.

The maximum number of function evaluations to find a solution of a certain quality was set to 1 000 000. In order to estimate the run-length and solution-quality distributions with acceptable accuracy, we ran the algorithms 100 times on each problem.

As we said before, we assigned different values to the algorithms' free parameters to measure their effects on the algorithms' final performance. These parameters were the number of particles and the swarm's population topology. We tried three different population sizes: 20, 40 and 60 particles, and three topologies: fully connected topology, ring topology and a square topology. The fully connected and ring topologies were described in Chapter 2. Here we will describe the square topology and the specific configurations we used.

The square topology can be thought to be a graph in which each node is connected to four neighbors forming a lattice with periodic boundaries. Figure 4.11 shows an example of a 5×4 square topology.

Square topologies can have different configurations for the same number of particles. For example, a population of 40 particles can be arranged in configurations of 5×8 or 4×10 particles. It is not clear which one yields the best results, but trying to be consistent with the idea of a "square" topology, we tried to keep the lattice as balanced as possible. The configurations that we used for 20, 40 and 60 particles were 5×4, 5×8 and 6×10 respectively.

Before presenting our results, it is important to note that we verified that our implementations could find similar results to those reported by their original

Table 4.1: Algorithms' fixed parameter settings.

Algorithm	Settings
Canonical Particle Swarm Optimizer	Acceleration coefficients φ_1 and φ_2 are set to 2.05. The constriction factor χ is set to 0.729. The maximum velocity V_{max} is clamped to $\pm X_{max}$, where X_{max} is the maximum of the search range.
Time-Decreasing Inertia Weight Particle Swarm Optimizer	Acceleration coefficients φ_1 and φ_2 are set to 2.0. The inertia weight decreases linearly from an initial value of 0.9 to a final one of 0.4. The final value is reached at the end of the run. The maximum velocity V_{max} is clamped to $\pm X_{max}$.
Time-Increasing Inertia Weight Particle Swarm Optimizer	Acceleration coefficients φ_1 and φ_2 are set to 2.0. The inertia weight value is increased linearly from an initial value of 0.4 to a final one of 0.9. The final value is reached at the end of the run. The maximum velocity V_{max} is clamped to $\pm X_{max}$.
Stochastic Inertia Weight Particle Swarm Optimizer	Acceleration coefficients φ_1 and φ_2 are set to 2.05. The inertia weight value is drawn from a uniform distribution in the range $[0.5, 1.0]$. The maximum velocityV_{max} is clamped to $\pm X_{max}$.
Fully Informed Particle Swarm Optimizer	φ (i.e., the sum of the acceleration coefficients) is equal to 4.1 and the constriction factor χ is set to 0.729. The maximum velocity V_{max} is clamped to $\pm X_{max}$.
Self-Organizing Hierarchical Particle Swarm Optimizer with Time-varying Acceleration Coefficients	φ_1 is linearly decreased from 2.5 to 0.5 and φ_2 is increased from 0.5 to 2.5. The reinitialization velocity is linearly decreased from V_{max} at the beginning of the run to $0.1 \cdot V_{max}$ at the end. The maximum velocity V_{max} is clamped to $\pm X_{max}$.
Adaptive Hierarchical Particle Swarm Optimizer	Acceleration coefficients φ_1 and φ_2 are set to 2.05. The constriction factor χ is set to 0.729. The initial branching factor is set to 20, and parameters d_{min}, f_{adapt}, and k_{adapt} were set to 2, $1000 * m$, and 3 respectively, where m is the number of particles.

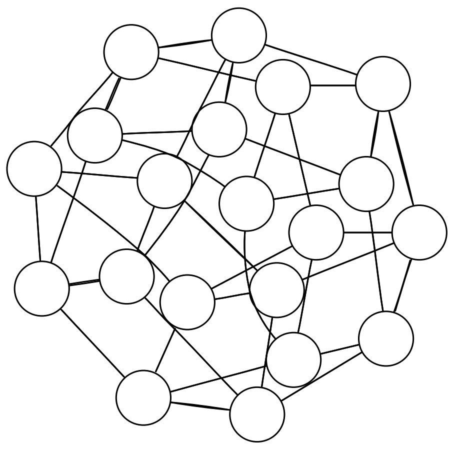


Figure 4.11: Example of a 5×4 square topology. Each particle is connected to four neighbors and the lattice thus formed has periodic boundaries.

authors. The interested reader is referred to the following URL to access the validation data: http://iridia.ulb.ac.be/supp/IridiaSupp2006-003/index.html.

4.2 Evaluation of Performance using Run-Length and Solution-Quality Distributions

In this section, we present the empirical performance evaluation of the particle swarm optimizers presented in Chapter 2. In the figures, they are referenced in the following way: the canonical particle swarm is referred to simply by *Canonical*, the increasing inertia weight particle swarm is referred to as *Increasing-IW*, the decreasing inertia weight particle swarm is labeled as *Decreasing-IW*, the stochastic inertia weight particle swarm is referred to as *Stochastic-IW*, the fully informed particle swarm is labeled as *FIPS*, the hierarchical particle swarm with time-varying acceleration coefficients is referred to as *HPSOTVAC*, and the adaptive hierarchical particle swarm is labeled *AHPSO*.

The comparison of the particle swarm optimizers' performance is carried out using the methodology presented in the previous chapter. The run-length and solution-quality distributions we discuss in this section were obtained using swarms of 20 and 60 particles. We focus our attention to the distributions obtained on four benchmark functions only (those that offered the most varied results). The run-length and solution-quality distributions presented in this section correspond to a specific solution quality and a certain maximum number of function evaluations. The complete set of results are available online at http://iridia.ulb.ac.be/supp/IridiaSupp2006-005/index.html.

The results obtained by AHPSO are compared with those of the other algorithms irrespective the specific population topology they used. That is, the results obtained by AHPSO are the same across topologies. We made this decision because AHPSO does not use a fixed topology (like the rest of PSOs), but it adapts it during the optimization process. This permits the measurement of the effects of an adaptive topology approach.

4.2.1 Particle Swarm Optimizers' Performance on the Ackley Function

Figure 4.12 shows the run-length distributions obtained by the seven particle swarm optimizers included in our comparison on the Ackley function. The solution quality requirement for these distributions was of 0.01% away from the global optimum. In absolute values (see Equation A.2), this solution quality corresponds to a solution whose objective function value is less than or equal to -139.996. The biased value of the global optimum is -140 as shown in Appendix A.

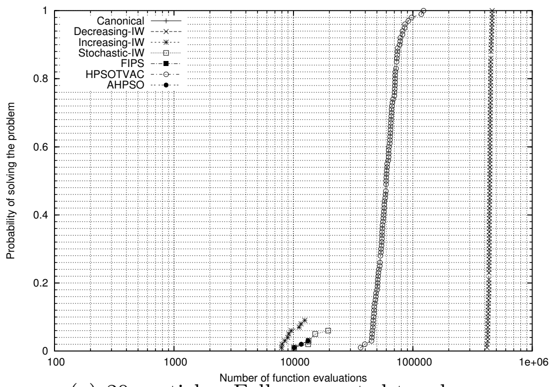
The only variants that (regardless the number of particles or the population topology) can find consistently the demanded solution quality are the decreasing inertia weight PSO and HPSOTVAC. Of these two variants, HPSOTVAC is much faster than the time decreasing inertia weight PSO (with the inertia weight decreasing schedule mentioned in Table 4.1). This statement does not even need to be based on some statistical test. The closest difference between the fastest and slowest runs of these two variants is of approximately 300 000 function evaluations (see Figure 4.12(d)).

Both, the number of particles and the population topology, have a strong effect on the algorithms' run-length distributions. Increasing the number of particles alleviates, in some degree, the algorithms' stagnating behavior. The same is true when we change from the fully connected topology to the ring or square topology, but most notably to the ring topology. Different algorithms are sensitive to these changes in different degrees. For example, the fully informed particle swarm is extremely sensitive to a change from a fully connected topology to a ring topology. In the first case, it fails completely, while in the second case, it is the fastest algorithm with a relatively high probability of success. This is also confirmed in Figure 4.14(b), which shows the median solution quality development over time for 20 particles and the ring topology.

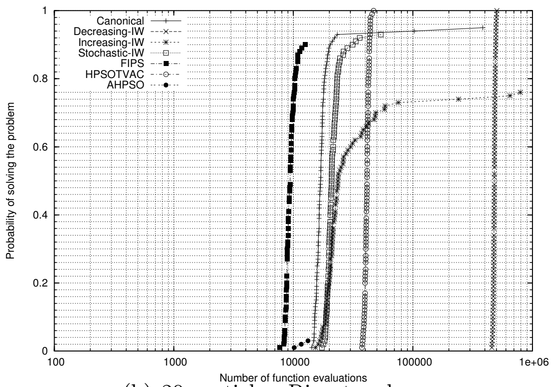
Figure 4.13 shows the solution-quality distributions obtained by the evaluated PSOs. These distributions correspond to 100 000 objective function evaluations. The solution quality range is set to the interval $[0.01, 10]$% or in absolute values, $[-139.9996, -136]$.

Since these plots correspond to 100 000 objective function evaluations, the decreasing inertia weight PSO shows a very poor performance. We know, from the inspection of the run-length distributions shown before, that after 1 000 000 function evaluations this variant finds a solution quality of 0.01% with probability equal to 1.0. Therefore, it finds solutions with deviations from the optimum greater than 0.01% with an equal probability.

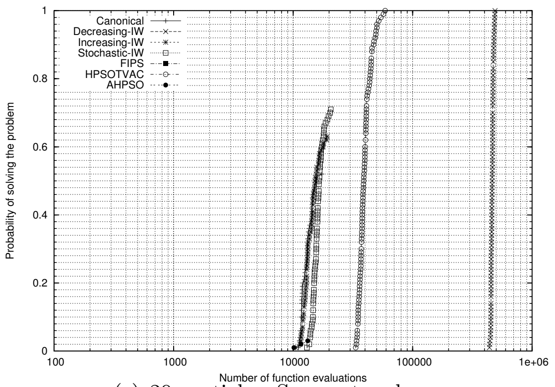
At solution qualities between 0.6% and 1.0% away from the optimum, almost all algorithms tend to get trapped somewhere in the search space. With 20 particles and the fully connected topology, almost all algorithms get stuck with a low probability of escaping from this region (see Figure 4.13(a)). This situation improves when the algorithms used either more particles, or the ring topology (see Figures 4.13(b) and 4.12(e)).



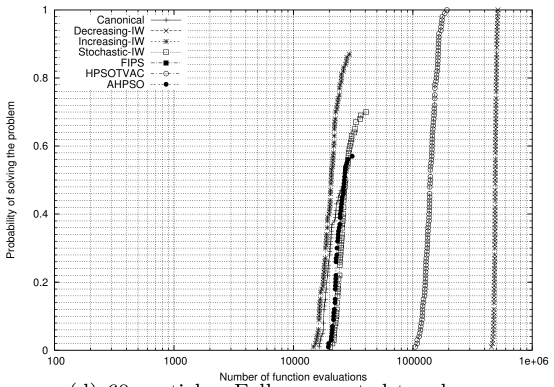
(a) 20 particles, Fully connected topology



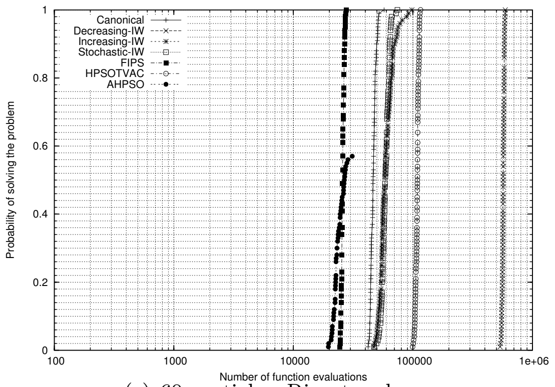
(b) 20 particles, Ring topology



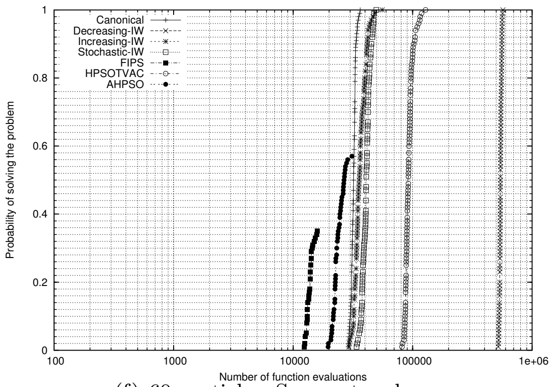
(c) 20 particles, Square topology



(d) 60 particles, Fully connected topology

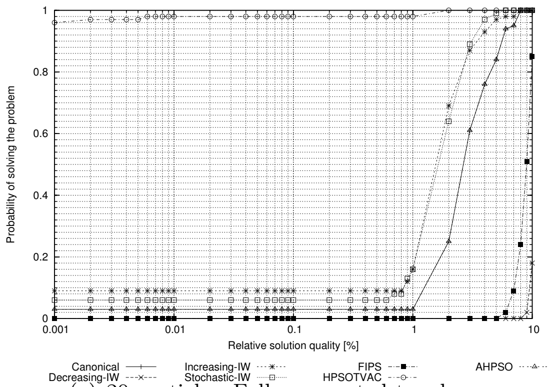


(e) 60 particles, Ring topology

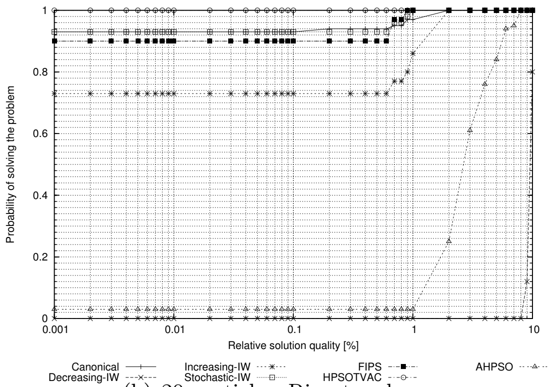


(f) 60 particles, Square topology

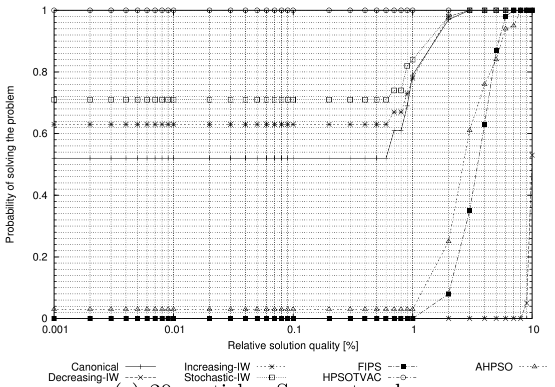
Figure 4.12: Run-Length Distributions on the Ackley Function. The solution quality demanded is of 0.01% away from the global optimum. Figures (a), (b), and (c) show the run-length distributions obtained with 20 particles. Figures (d), (e), and (f) show the run-length distributions obtained with 60 particles. Columns show the effect of using different population topologies.



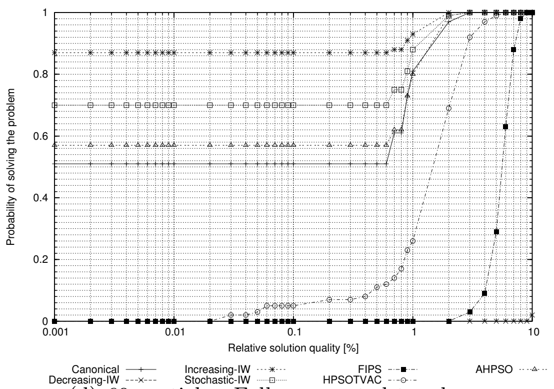
(a) 20 particles, Fully connected topology



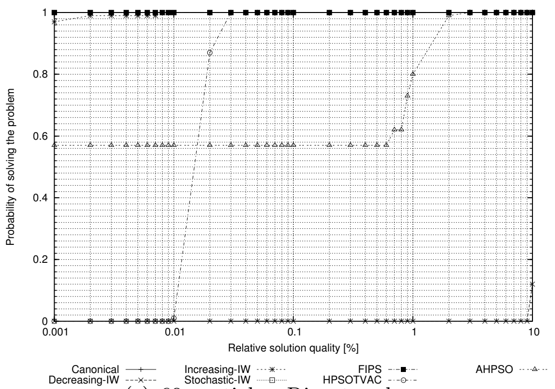
(b) 20 particles, Ring topology



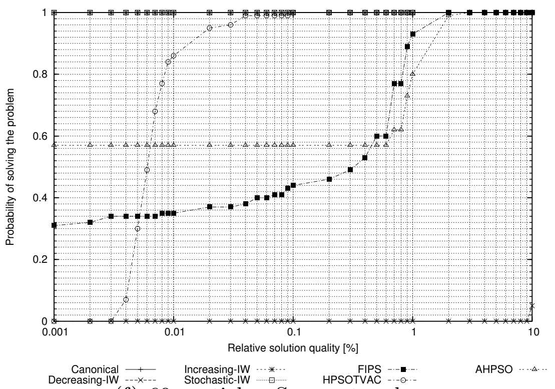
(c) 20 particles, Square topology



(d) 60 particles, Fully connected topology

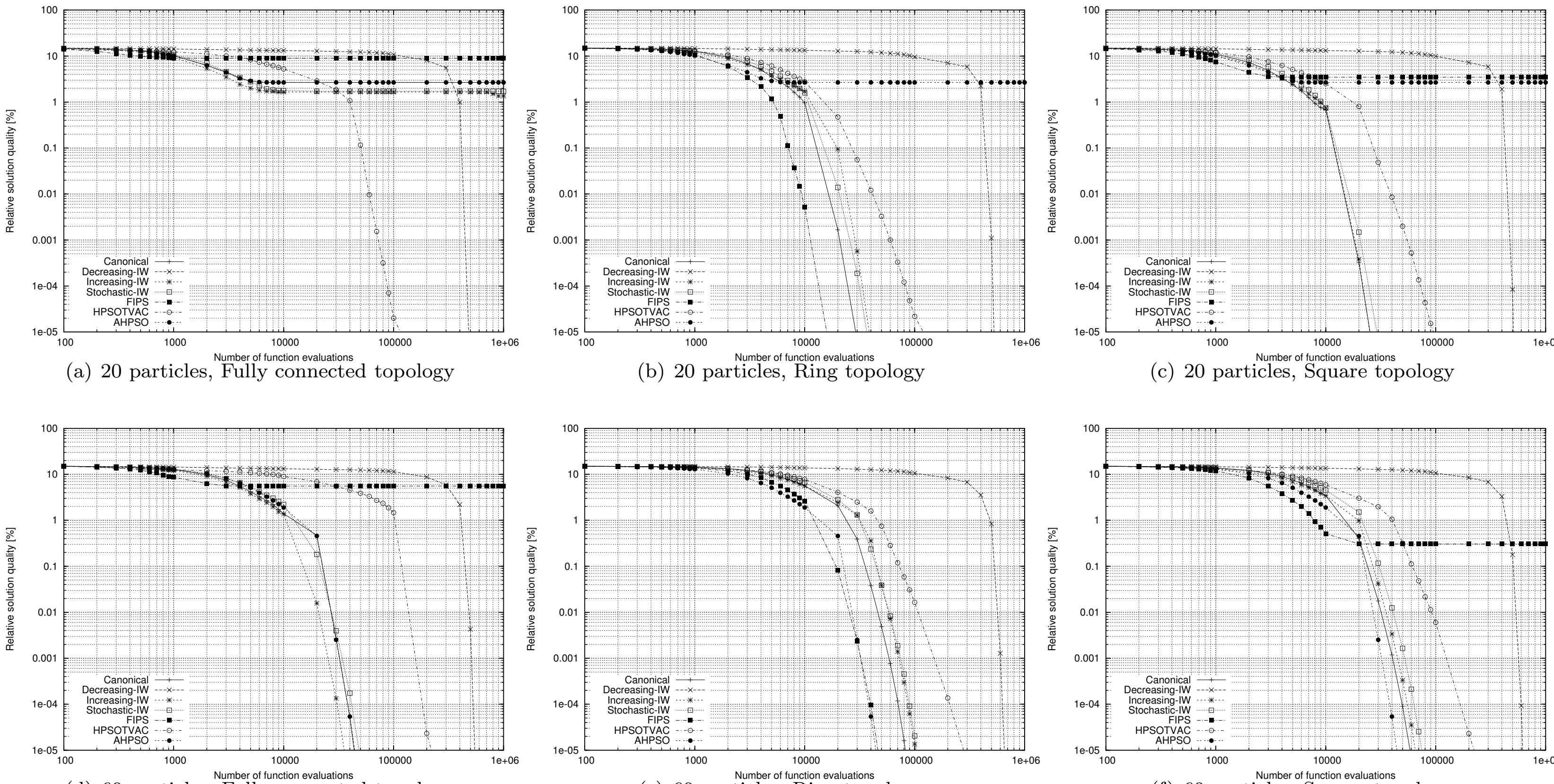


(e) 60 particles, Ring topology



(f) 60 particles, Square topology

Figure 4.13: Solution-Quality Distributions on the Ackley Function. These results were obtained after 100 000 objective function evaluations. Figures (a), (b), and (c) show the solution-quality distributions obtained with 20 particles. Figures (d), (e), and (f) show the solution-quality distributions obtained with 60 particles. Columns show the effect of using different population topologies.



(a) 20 particles, Fully connected topology

(b) 20 particles, Ring topology

(c) 20 particles, Square topology

(d) 60 particles, Fully connected topology

(e) 60 particles, Ring topology

(f) 60 particles, Square topology

Figure 4.14: Solution quality development over time on the Ackley function. Figures (a), (b), and (c) show the solution quality development over time that is obtained with 20 particles. Figures (d), (e), and (f) show the solution quality development over time that is obtained obtained with 60 particles. Columns show the effect of using different population topologies. These results are based on medians.

It is interesting to note that the HPSOTVAC variant is negatively affected with an increase in the size of the population. When HPSOTVAC ran with 20 particles, the probability was near 1.0 independently of the topology being used. With 60 particles, the probability of finding a solution with a percentage deviation lower than 0.003% is 0.0 (compare Figure 4.13(a) with Figure 4.12(d), and so on). All other PSOs (except the decreasing inertia weight variant, which has no significative change) benefit from an increase in the population size.

Figure 4.14 shows the solution-quality development over time plots of the evaluated PSOs on the Ackley function. These plots depict median values.

The fastest PSO variant in finding high quality solutions for the Ackley function is the fully informed particle swarm. This is only true when using 20 particles and the ring topology. However, we know from our previous observations, that it is not the most reliable algorithm. However, the fastest algorithm in finding a solution quality 0.01% away from the optimum with a probability of 1.0, is still the FIPS variant but using 60 particles and the ring topology (see Figure 4.12(e)).

4.2.2 Particle Swarm Optimizers' Performance on the Easom Function

Figure 4.15 shows the empirical run-length distributions of the compared PSOs, obtained at a quality level of 0.0001% away from the optimum, on the Easom function. In absolute values, this solution quality is equivalent to an objective function value of -0.999999. The optimum has a function value of -1.0.

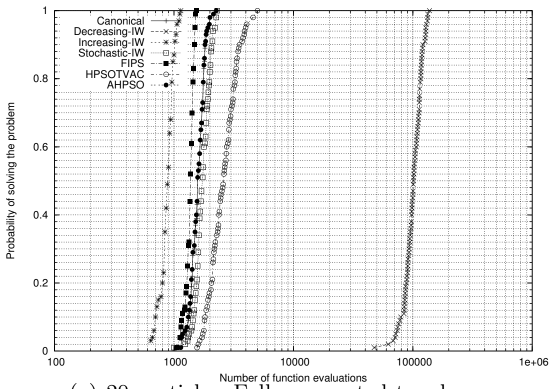
All algorithms are capable of finding a solution of this required solution quality. Given the characteristics of the Easom function, this means that none of the tested algorithms gets trapped when there is a large plateau and just a small area with the global optimum. This conclusion is valid for the two-dimensional Easom function. In preliminary experiments, a high-dimensional Easom function proved to be very difficult for all the tested PSOs.

The decreasing inertia weight variant is the variant with the slowest convergence towards the required solution quality: approximately one order of magnitude slower than the slowest of the other six PSOs (median speed). This variant is also the only one with a slowly-increasing probability of success, most notably when it uses 60 particles and a square topology (see Figure 4.15(f)). This might be caused by some "lucky" runs which found the solution region early while they were still in exploratory behavior.

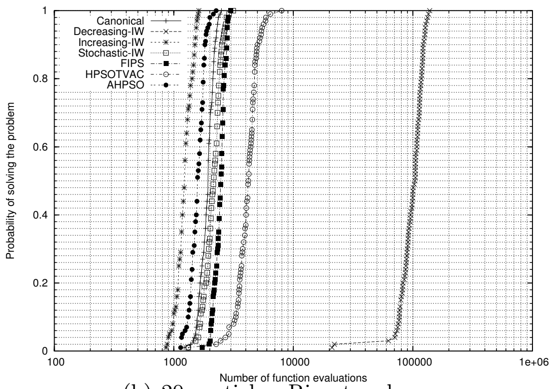
The effect of the population topology is not so strong as is the effect of the number of particles. Increasing the number of particles results in an overall slowdown. As explained in the previous chapter, this can be seen as run-length distributions shifted to the right (compare Figure 4.15(a) with Figure 4.15(d), and so on).

A change in the population topology seems to just change the relative performance of the canonical, FIPS and stochastic inertia weight variants. AHPSO is, by definition, not affected by a change in the population topology. In all cases, irrespective to the topology or the number of particles, the increasing inertia weight is the fastest particle swarm optimizer to find the required solution quality.

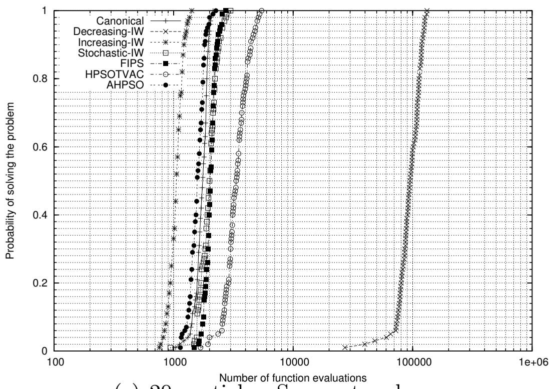
Figure 4.16 shows the solution-quality distributions, at 1 000 objective function evaluations, obtained by the evaluated PSOs on the Easom function.



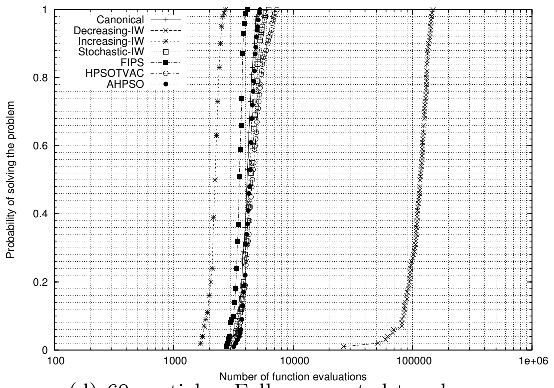
(a) 20 particles, Fully connected topology



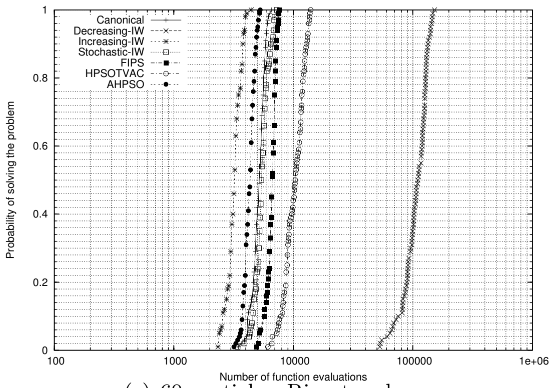
(b) 20 particles, Ring topology



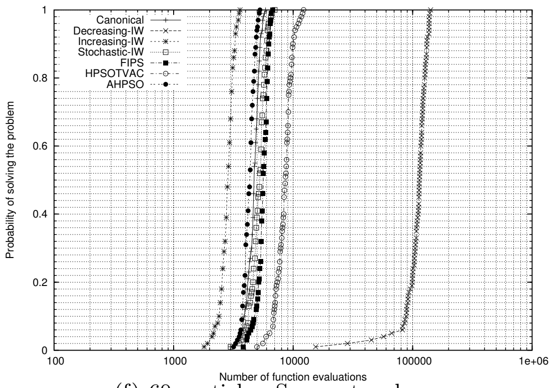
(c) 20 particles, Square topology



(d) 60 particles, Fully connected topology

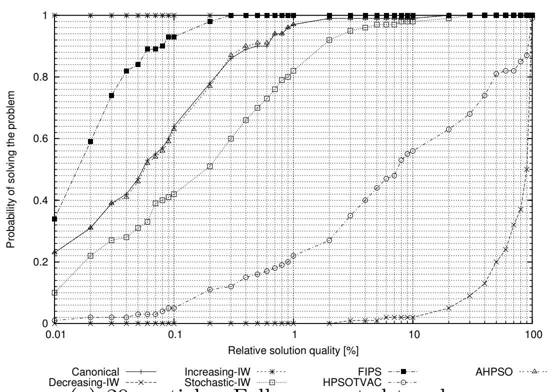


(e) 60 particles, Ring topology

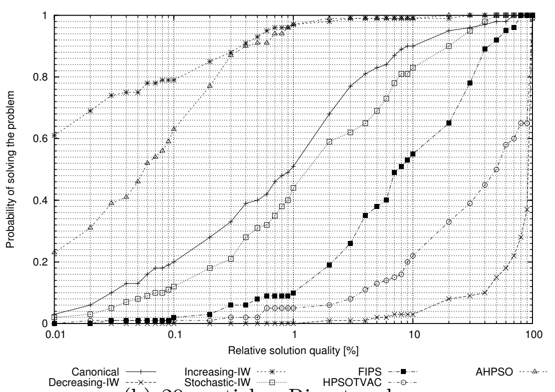


(f) 60 particles, Square topology

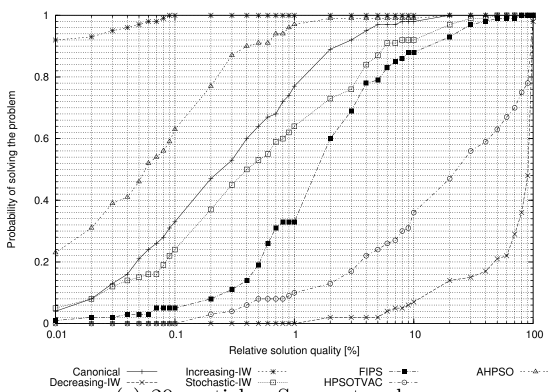
Figure 4.15: Run-Length Distributions on the Easom Function. The solution quality demanded is of 0.0001% away from the global optimum. Figures (a), (b), and (c) show the run-length distributions obtained with 20 particles. Figures (d), (e), and (f) show the run-length distributions obtained with 60 particles. Columns show the effect of using different population topologies.



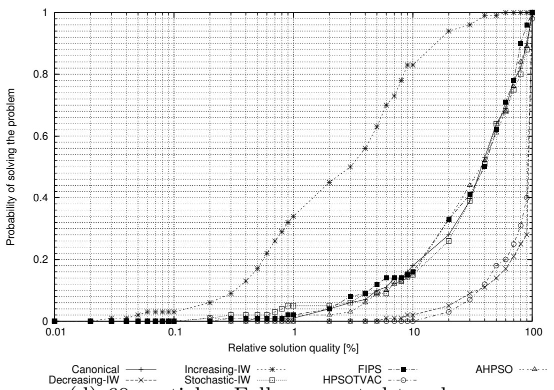
(a) 20 particles, Fully connected topology



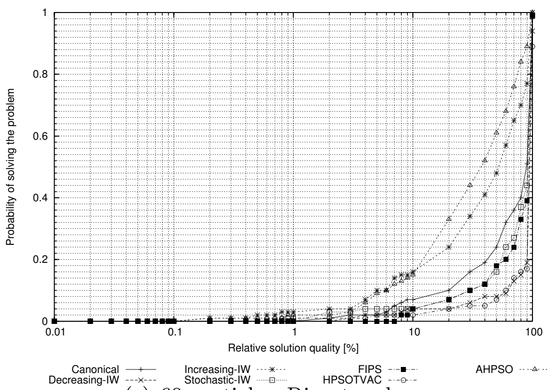
(b) 20 particles, Ring topology



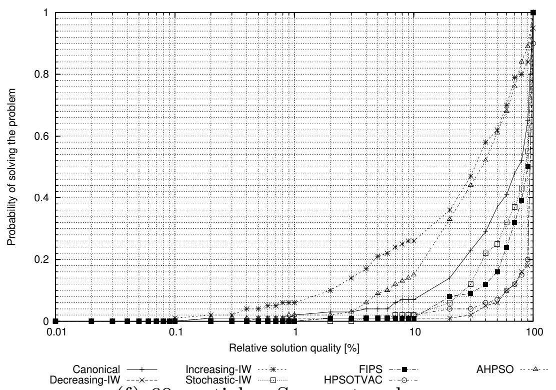
(c) 20 particles, Square topology



(d) 60 particles, Fully connected topology

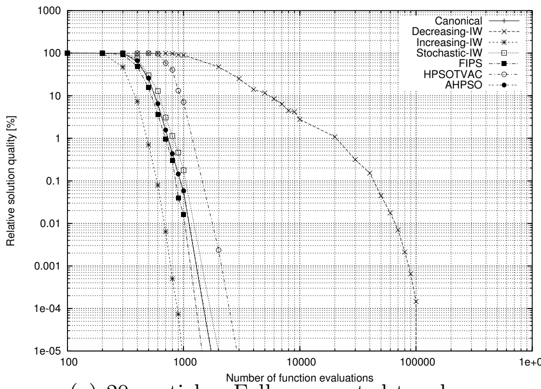


(e) 60 particles, Ring topology

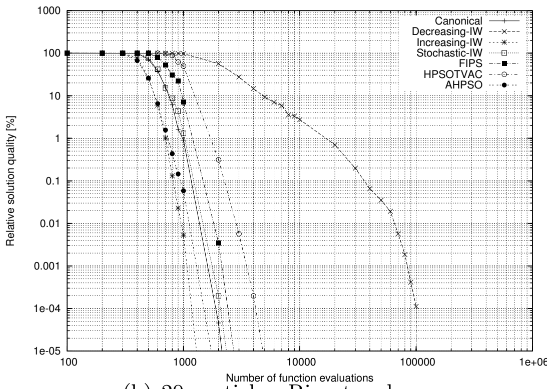


(f) 60 particles, Square topology

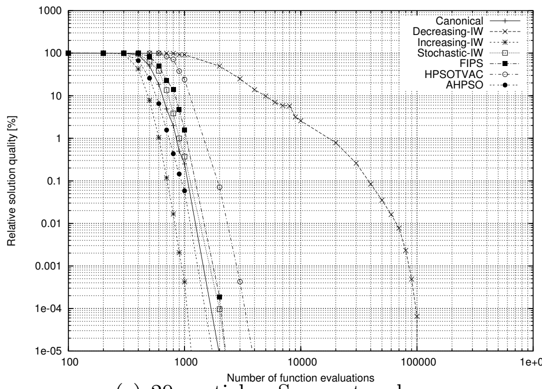
Figure 4.16: Solution-Quality Distributions on the Easom Function. These results were obtained after 1000 objective function evaluations. Figures (a), (b), and (c) show the solution-quality distributions obtained with 20 particles. Figures (d), (e), and (f) show the solution-quality distributions obtained with 60 particles. Columns show the effect of using different population topologies.



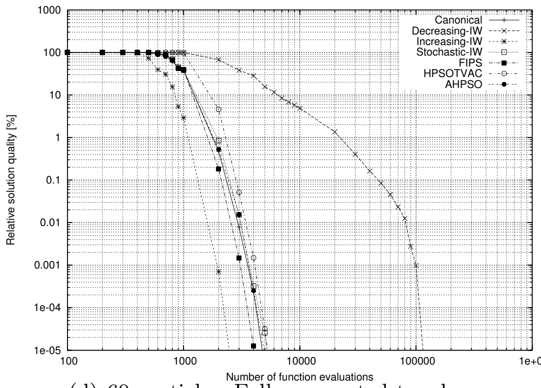
(a) 20 particles, Fully connected topology



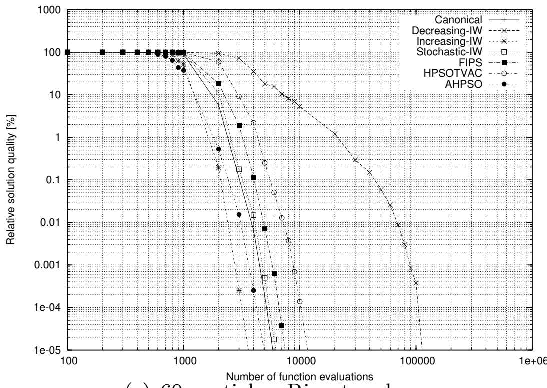
(b) 20 particles, Ring topology



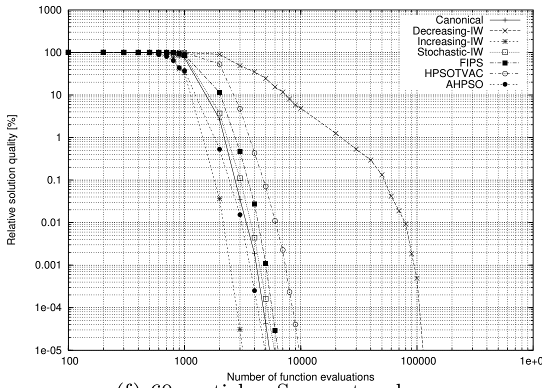
(c) 20 particles, Square topology



(d) 60 particles, Fully connected topology



(e) 60 particles, Ring topology



(f) 60 particles, Square topology

Figure 4.17: Solution quality development over time on the Easom function. Figures (a), (b), and (c) show the solution quality development over time that is obtained with 20 particles. Figures (d), (e), and (f) show the solution quality development over time that is obtained obtained with 60 particles. Columns show the effect of using different population topologies. These results are based on medians.

The solution quality range is set to the interval $[100, 0.01]\%$ or in absolute values, $[0.0, -0.9999]$. Given the fact that the Easom function cannot take a value greater than 0.0, all algorithms have a probability of 1.0 of finding this solution quality level.

The increasing inertia weight variant finds good solution qualities with high probabilities after just 1000 function evaluations. Using the fully connected topology gives this variant a special "greediness" that allows it to find higher quality solutions than the rest of the PSOs. FIPS is also affected in the same way although not very much when using 60 particles. AHPSO performs very well although its adaptation from a highly connected topology to a loosely connected one makes it the second best performer in this function.

In Figure 4.17, we show the median solution quality development over time, of the evaluated PSOs, on the Easom function. From this figure, a strong convergent behavior is beneficial for having a good performance in this function. It seems that big step sizes, favored by an explorative behavior, cause the algorithms to miss the region where the optimum is. Drastically reducing particles velocities seems to be a good strategy to force them to take short step sizes and this increases their probability of success.

4.2.3 Particle Swarm Optimizers' Performance on the Griewank Function

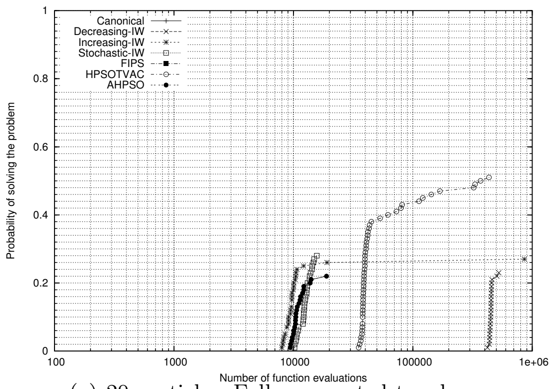
The empirical run-length distributions of the tested PSOs on the Griewank function are shown in Figure 4.18. The required solution quality is of 0.001% away from the optimum. This solution quality is equivalent to an objective function value of -179.9982. The optimum has a function value of -180.0.

No algorithm was capable of finding a solution of the required quality with a probability of 1.0 with a population size of 20 particles. In fact, it is the only case in our test suite in which this phenomenon is present. All algorithms get trapped in the search space. With 60 particles, only FIPS finds the required solution quality with probability 1.0, while the canonical PSO and HPSOTVAC reach a probability of 0.99.

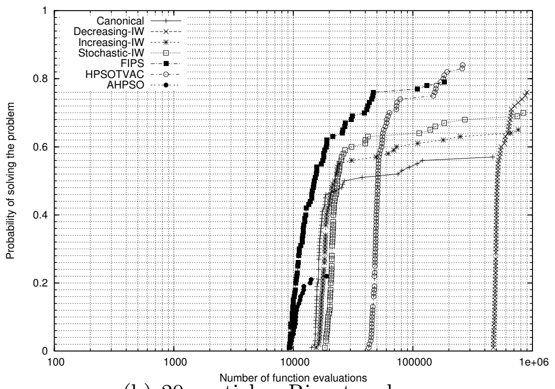
The interaction between the population topology and the number of particles is very clear. All algorithms (except AHPSO, due to its adaptive topology mechanism) benefit from a topology that facilitates an explorative behavior. The explorative behavior of the algorithms is also boosted with a large population size. This gives as a result the best performance (in terms of reliability) when algorithms used 60 particles and the ring topology (see Figure 4.18(e)).

Figure 4.19 shows the solution-quality distributions of the tested PSOs at 1 000 000 function evaluations. All algorithms suffer from severe stagnation, specially when they reach a solution quality of about 0.003%. HPSOTVAC dominates the rest of the algorithms when using 20 particles, that is, it is finds high quality solutions with greater probabilities than the others, regardless the population topology. It is also the dominant variant when using 60 particles, except when using the ring topology, in which it was FIPS the dominant algorithm.

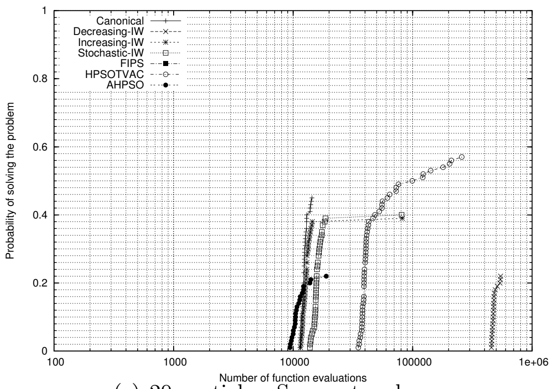
Figure 4.20 shows the median quality development over time. Even though it is not the most reliable algorithm, the fastest algorithm is FIPS with 20 particles and the ring topology. The slowest algorithm is the decreasing inertia weight variant.



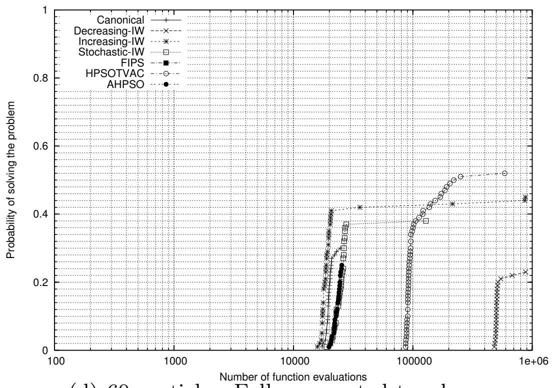
(a) 20 particles, Fully connected topology



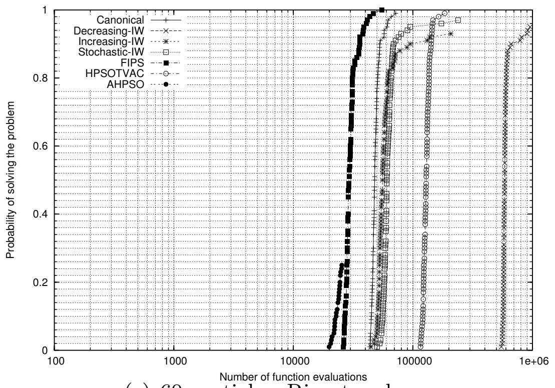
(b) 20 particles, Ring topology



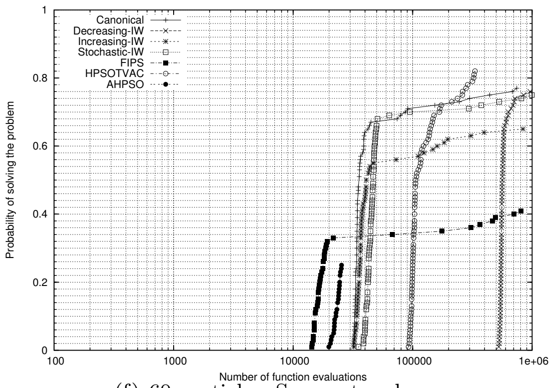
(c) 20 particles, Square topology



(d) 60 particles, Fully connected topology

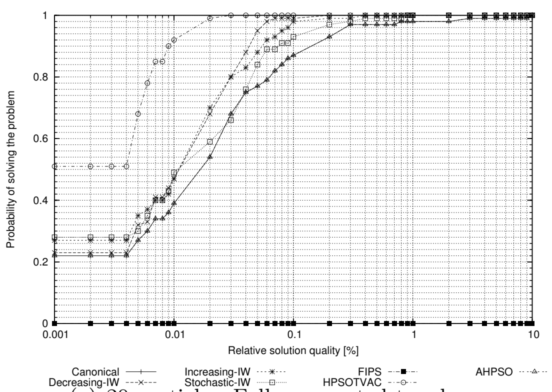


(e) 60 particles, Ring topology

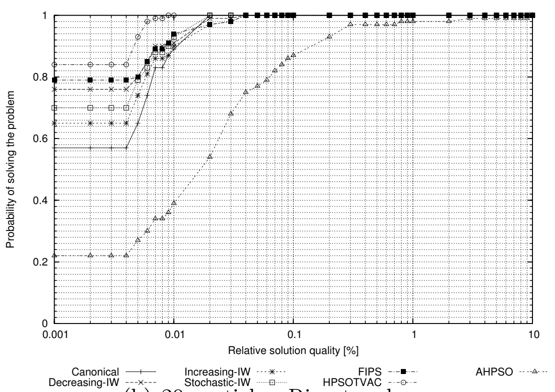


(f) 60 particles, Square topology

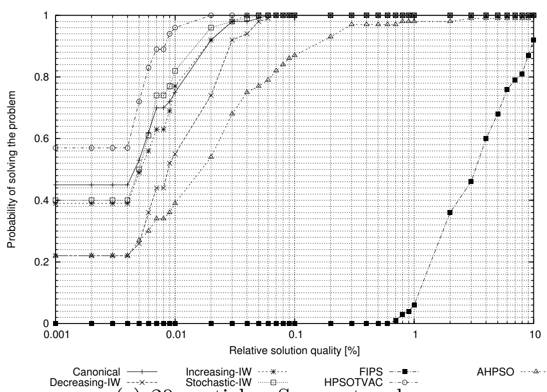
Figure 4.18: Run-Length Distributions on the Griewank Function. The solution quality demanded is of 0.001% away from the global optimum. Figures (a), (b), and (c) show the run-length distributions obtained with 20 particles. Figures (d), (e), and (f) show the run-length distributions obtained with 60 particles. Columns show the effect of using different population topologies.



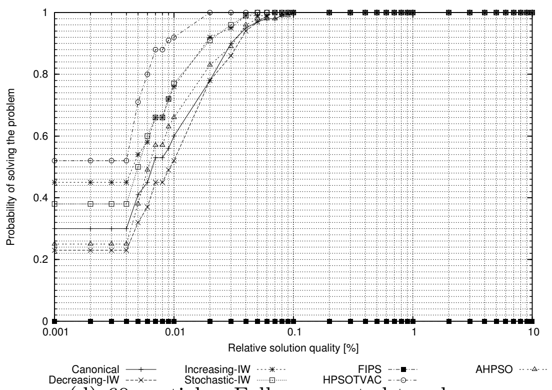
(a) 20 particles, Fully connected topology



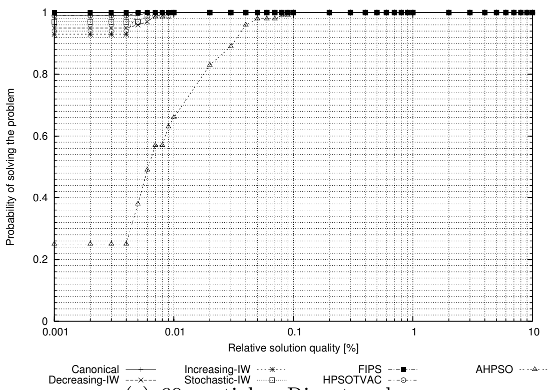
(b) 20 particles, Ring topology



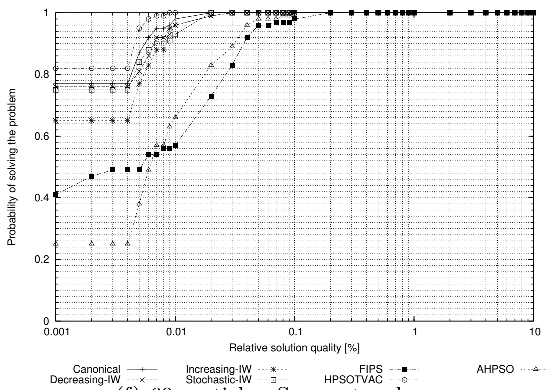
(c) 20 particles, Square topology



(d) 60 particles, Fully connected topology

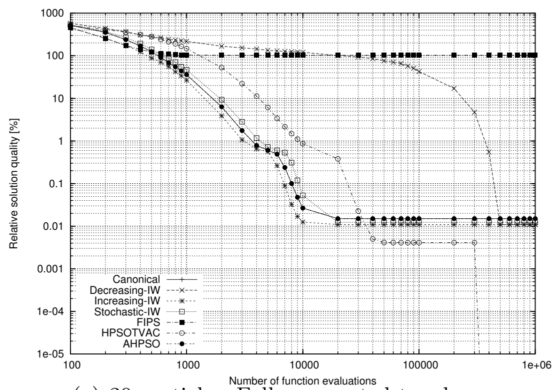


(e) 60 particles, Ring topology

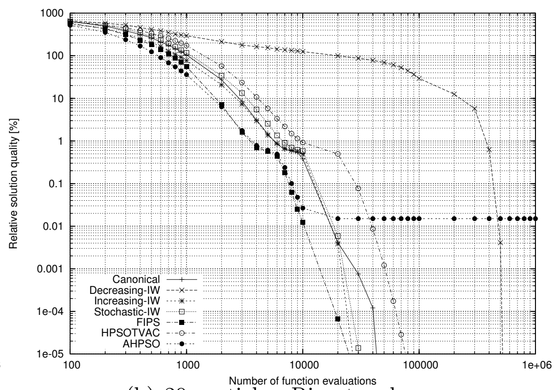


(f) 60 particles, Square topology

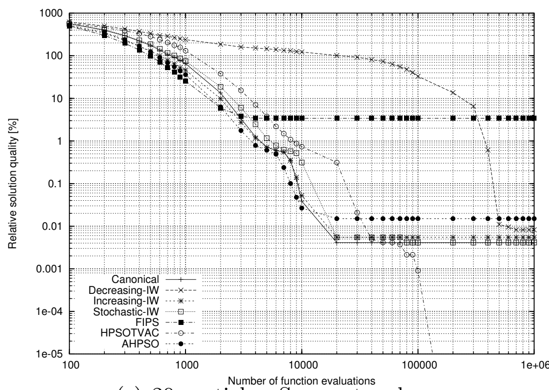
Figure 4.19: Solution-Quality Distributions on the Griewank Function. These results were obtained after 1 000 000 objective function evaluations. Figures (a), (b), and (c) show the solution-quality distributions obtained with 20 particles. Figures (d), (e), and (f) show the solution-quality distributions obtained with 60 particles. Columns show the effect of using different population topologies.



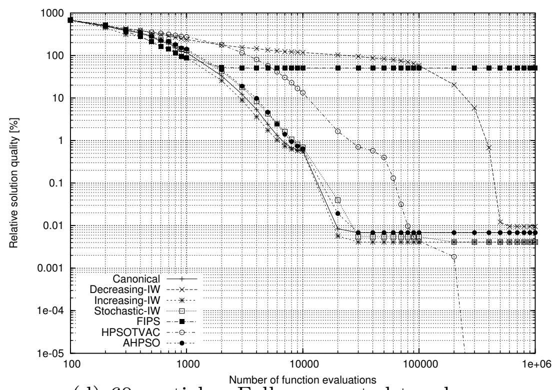
(a) 20 particles, Fully connected topology



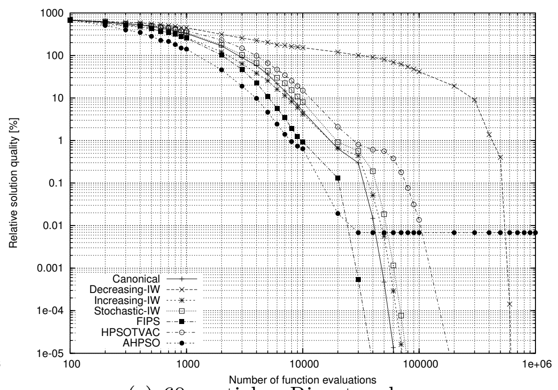
(b) 20 particles, Ring topology



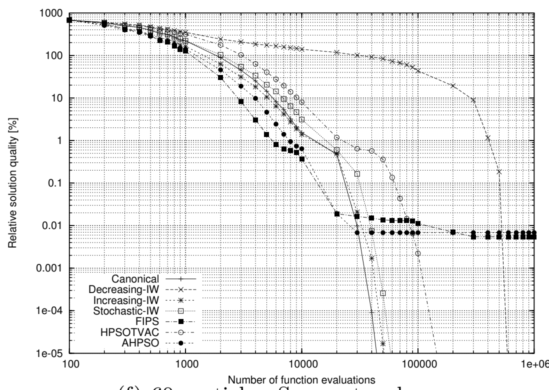
(c) 20 particles, Square topology



(d) 60 particles, Fully connected topology



(e) 60 particles, Ring topology



(f) 60 particles, Square topology

Figure 4.20: Solution quality development over time on the Griewank function. Figures (a), (b), and (c) show the solution quality development over time that is obtained with 20 particles. Figures (d), (e), and (f) show the solution quality development over time that is obtained obtained with 60 particles. Columns show the effect of using different population topologies. These results are based on medians.

4.2.4 Particle Swarm Optimizers' Performance on the Rastrigin Function

Figure 4.21 shows the run-length distributions of the tested PSOs on the Rastrigin function at a solution quality level of 20.0% away from the optimum. This solution quality requirement is equivalent to a solution whose objective function value is -264.0. The optimum has a function value of -330.0.

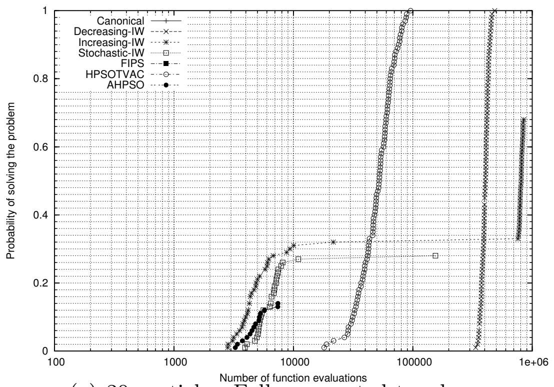
Only two variants are capable of achieving the required solution quality with probability 1.0, in a consistent way, irrespective to the population size or topology. These algorithms are the decreasing inertia weight variant and HPSOTVAC. The slowest of these two variants is the decreasing inertia weight variant.

On this function, the population topology that yields the best overall performance is the square topology. With this topology, almost all algorithms increase the probability of finding the required solution quality. The exceptions are AH-PSO and FIPS. FIPS achieves its best performance with the ring topology. An interesting case is the one of the increasing inertia weight variant. The probability of finding a solution of the required solution abruptly increases around 800 000 function evaluations. At this point in time, the inertia weight takes a value of 0.8 according to Equation 2.7 and the settings used. This abrupt change indicates that the probability density function is multimodal with a high peak near this region. A possible explanation of this behavior is that this variant gets trapped in some local minimum very rapidly, and only when the particles' velocity is increased beyond some threshold, the swarm is capable of escaping from it.

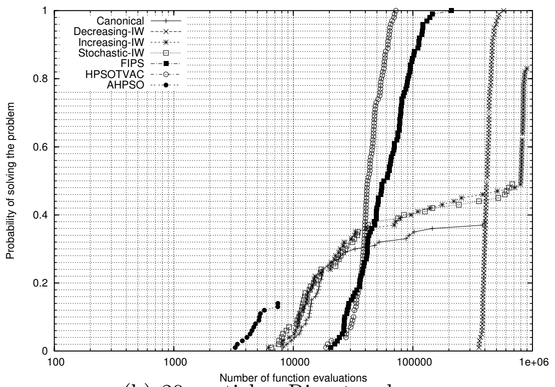
Figure 4.22 shows the solution-quality distributions of all the tested PSOs at 1 000 000 function evaluations. The only variant that is capable of finding high quality solutions (e.g., 1% away from the optimum), is HPSOTVAC. The second best algorithm in this respect is the time decreasing inertia weight variant.

Increasing the number of particles does not have a big impact on most of the algorithms' performance. The exception is HPSOTVAC. When HPSOTVAC uses 60 particles, it gets trapped in a region whose solution quality level is around 0.3% away from the optimum (absolute value of -329.01). The probability of escaping from this region depends on the population topology, being the fully connected topology the one that shows the highest one.

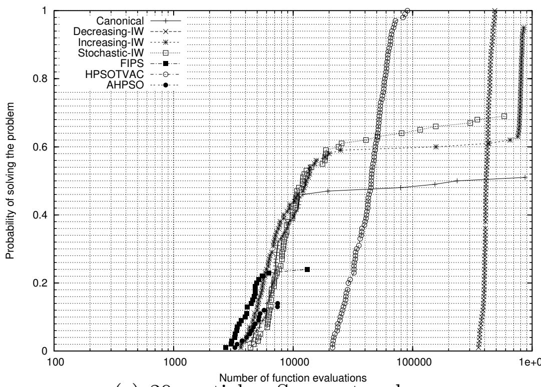
In Figure 4.23, we show the solution quality development over time graphs of the studied PSOs. With the fully connected topology, FIPS starts improving the quality of the found solutions quickly but stagnates very soon too. The increasing inertia weight improves the solution a bit slower that FIPS and stagnates later at around 20% away from the optimum. HPSOTVAC and the decreasing inertia weight variant are among the slowest variants up to 10 000 function evaluations. The shape of their curves, reveals that they are not too greedy at first but in the long run they are best performers.



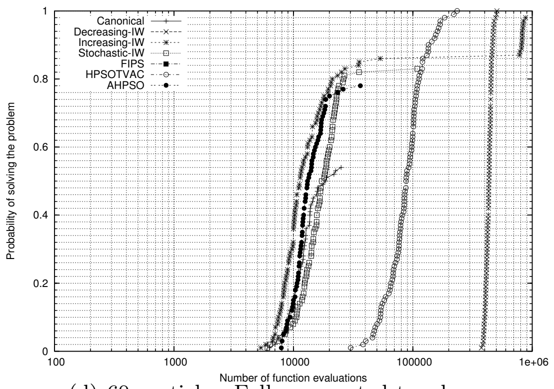
(a) 20 particles, Fully connected topology



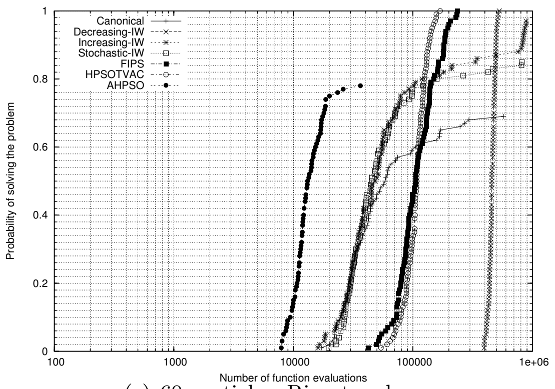
(b) 20 particles, Ring topology



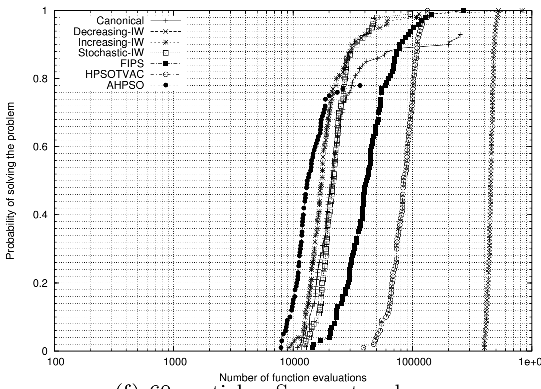
(c) 20 particles, Square topology



(d) 60 particles, Fully connected topology

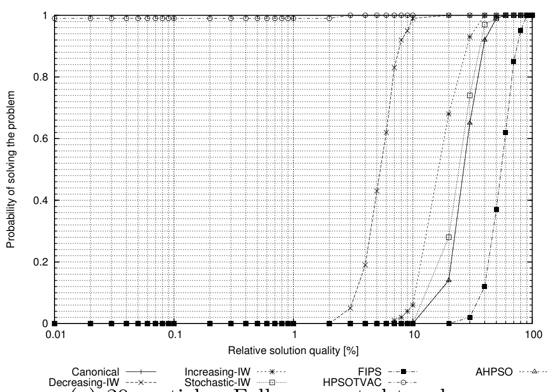


(e) 60 particles, Ring topology

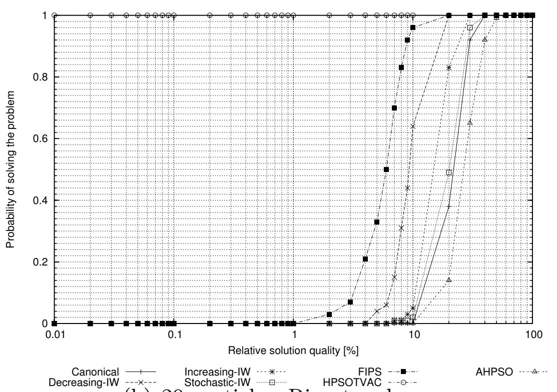


(f) 60 particles, Square topology

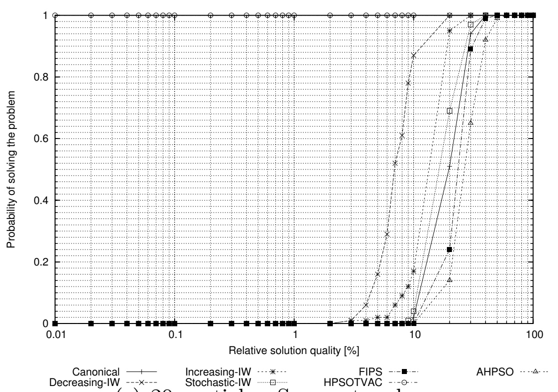
Figure 4.21: Run-Length Distributions on the Rastrigin Function. The solution quality demanded is of 20.0% away from the global optimum. Figures (a), (b), and (c) show the run-length distributions obtained with 20 particles. Figures (d), (e), and (f) show the run-length distributions obtained with 60 particles. Columns show the effect of using different population topologies.



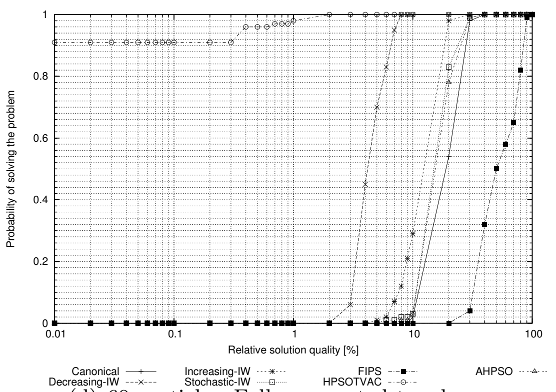
(a) 20 particles, Fully connected topology



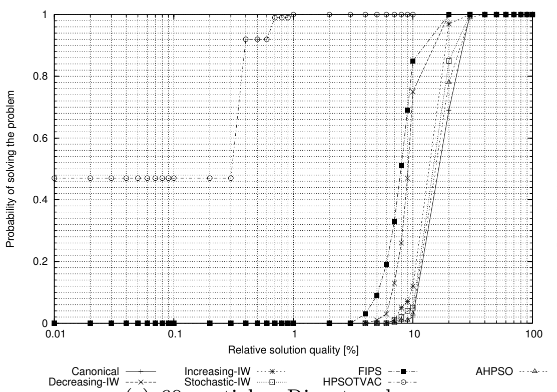
(b) 20 particles, Ring topology



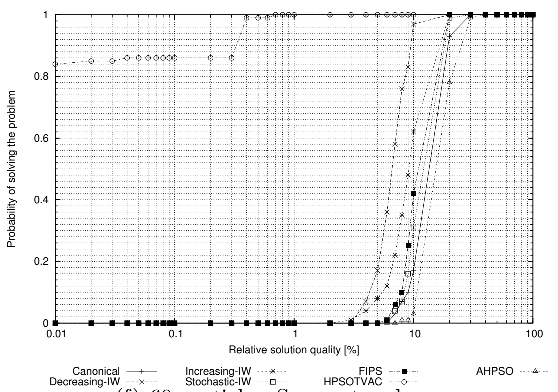
(c) 20 particles, Square topology



(d) 60 particles, Fully connected topology

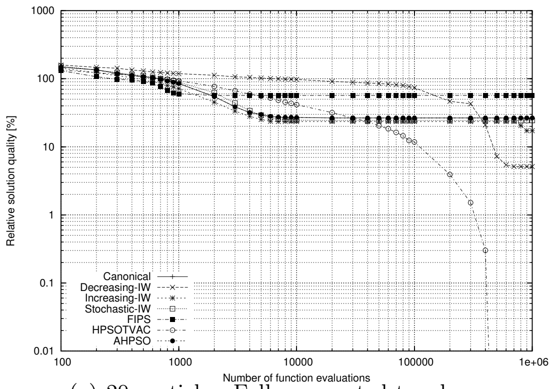


(e) 60 particles, Ring topology

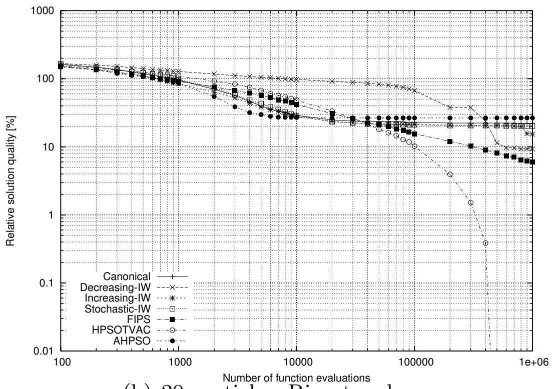


(f) 60 particles, Square topology

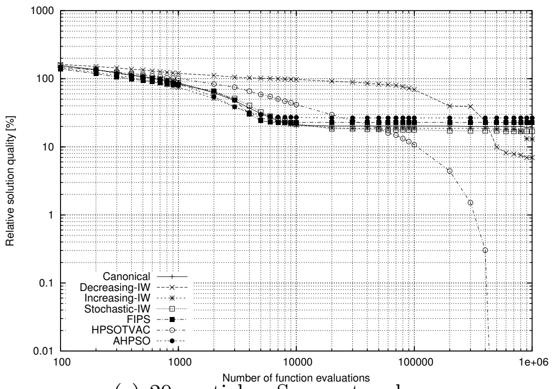
Figure 4.22: Solution-Quality Distributions on the Rastrigin Function. These results were obtained after 1 000 000 objective function evaluations. Figures (a), (b), and (c) show the solution-quality distributions obtained with 20 particles. Figures (d), (e), and (f) show the solution-quality distributions obtained with 60 particles. Columns show the effect of using different population topologies.



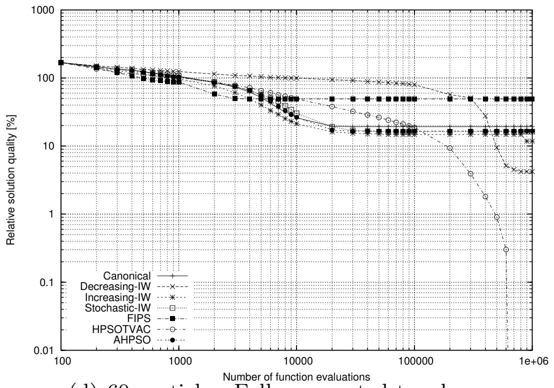
(a) 20 particles, Fully connected topology



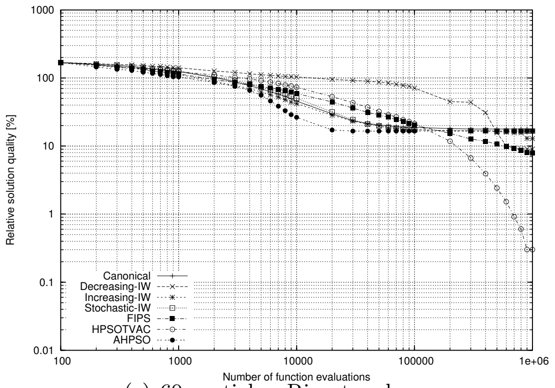
(b) 20 particles, Ring topology



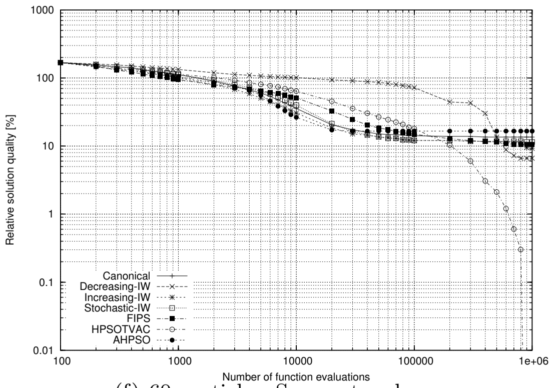
(c) 20 particles, Square topology



(d) 60 particles, Fully connected topology



(e) 60 particles, Ring topology



(f) 60 particles, Square topology

Figure 4.23: Solution quality development over time on the Rastrigin function. Figures (a), (b), and (c) show the solution quality development over time that is obtained with 20 particles. Figures (d), (e), and (f) show the solution quality development over time that is obtained obtained with 60 particles. Columns show the effect of using different population topologies. These results are based on medians.

4.3 Quantitative Evaluation of Particle Swarm Optimizers' Performance

In the previous section, we presented some of the run-length and solution quality distributions the compared particle swarm optimizers exhibit for certain solution qualities and computational effort limits, respectively. Even though they provide a wealth of information about the behavior and performance of the algorithms, we cannot use them directly to have a clear picture of the algorithms' relative performance across several benchmark functions. In order to do that, and to quantitatively describe the performance of stochastic optimization algorithms, one needs to focus on some specific descriptive statistics of the run-length and solution-quality distributions, such as their means, standard deviations, maximums, minimums, medians, and so on.

In this section we present an analysis based on the median solution quality achieved by an algorithm after some specific number of function evaluations. We decided to base our analysis on medians because, as seen in the run-length and solution-quality distributions, algorithms exhibit a high performance variability. An analysis based on medians is more robust and stable than an analysis based on means.

Our experimental design considers particle swarm optimizers with 20, 40 and 60 particles using fully connected. ring and square topologies. The set of benchmark functions is composed of 10 functions with different characteristics each. Additionally, we want to know the relative performance of these optimizers when they are given budgets of 1000, 10 000, 100 000 and 1 000 000 function evaluations. Thus, we have a factorial design with 3 primary factors: the tested algorithm (with 7 levels), the number of particles (with 3 levels), and the population topology (with 3 levels), and 2 blocking factors: the benchmark function (with 10 levels), and the budget (with 4 levels).

Our analysis is based only on the 40 particles case. It was a compromise that helped us focus our analysis to the effects of the population topology. The choice of 40 particles was made on the assumption that a medium-sized population is a sensible choice to balance fast convergence and high population diversity.

We begin the presentation of our analysis by describing the effects of different population topologies on the performance of the compared particle swarm optimizers. We also measure the sensitiveness of the different algorithms to a change in the population topology. We then determine the best performing particle swarm optimizers identifying, at the same time, the conditions and factors that make them work best.

4.3.1 Effects of Population Topologies on Particle Swarm Optimizers' Performance

Since no assumptions about normality or otherwise on the solution-quality distributions were justified, we ranked the algorithms' performance according to the median solution quality they were able to achieve achieved after a certain number of function evaluations. In the case two or more algorithms achieved the same solution quality value, that is, if there was a "tie", they were assigned an average rank. For example, if ranks 4, 5 and 6 have the same solution quality value, we give each observation the rank 5. The actual values of the medians are reported in Appendix B on Tables B.1, B.2 and B.3.

The tables shown below, report the ranks obtained by the compared particle swarm optimizers on every benchmark function. Each cell contains four entries which correspond to the rank obtained by the corresponding optimizer after 1 000, 10 000, 100 000 and 1 000 000 function evaluations, respectively. Benchmark functions are divided into four groups: high dimensional multimodal functions, high dimensional unimodal functions, low dimensional multimodal functions, and low dimensional unimodal functions. The rightmost column shows the average rank, obtained by the corresponding optimizer over all benchmark functions after a certain number of function evaluations (FES).

Table 4.2 shows the ranking across benchmark functions and function evaluations of the compared optimizers using a fully connected topology. The arrangement allows the reader to have an idea about the relative performance of each algorithm on every function over the whole optimization process. It is evident that there is no algorithm that dominates the others all the time on all benchmark functions. The only case in which one algorithm dominates the others is when solving the Rosenbrock function. In this case, the canonical particle swarm optimizer is ranked first all the time. The other extreme case, in which an algorithm is always ranked last, is the case of AHPSO solving the Salomon function. In all other cases, every algorithm is, at least, better than another one on some function at some point in time.

The decreasing inertia weight particle swarm is always ranked last, or second to last, during the first 100 000 function evaluations. It has a very slow convergence during the first phases, compared to the other PSOs. However, it achieves the best rank, on all the benchmark functions except on Griewank, Rastrigin (in which it is ranked second), and Rosenbrock problems, after 1 000 000 function evaluations.

The first places, at 1 000 function evaluations, are shared between the increasing inertia weight particle swarm and FIPS. With the exception of Schwefel, Salomon, and Rosenbrock functions, these two variants are always ranked first or second places. The situation changes completely for FIPS at 10 000 function evaluations. At this point, FIPS is already ranked the second worst. At 1 000 000 function evaluations, FIPS is the worst of all algorithms except in the Easom function, in which it shares the same rank with the other optimizers. On the contrary, the increasing inertia weight variant remains the top optimizer not only at 10 000 function evaluations, but also at 100 000. At 1 000 000 function evaluations it becomes the second best, behind the decreasing inertia weight particle swarm.

Similarly to the decreasing inertia weight particle swarm, HPSOTVAC improves its ranking after several function evaluations. In fact, these two optimizers share out the first ranks at 1 000 000 function evaluations, except in the Rosenbrock function. HPSOTVAC is the top optimizer for Griewank and Rastrigin functions.

Figure 4.24 shows the particle swarm optimizers' relative average ranking as a function of the number of function evaluations. In this figure, the interaction between the algorithms' ranking and the number of function evaluations, is easily visualized.

Table 4.2: Ranking of particle swarm optimizers based on the median solution quality achieved after a certain number of function evaluations. Fully connected topology case. Each cell contains the rank obtained by the corresponding optimizer after 1 000, 10 000, 100 000 and 1 000 000 function evaluations (FES).

	Fully connected topology case											
		High dimensional								Low dimensional		
		Multimodal						Unimodal		Multimodal	Unimodal	
	FES	Ackley	Step	Griewank	Rastrigin	Schwefel	Salomon	Sphere	Rosenbrock	Schaffer	Easom	Avg. Rank
Canonical	10^3	3	4	3	5	3	4	3	1	4	5	3.50
	10^4	3	2.5	2	3	4	1	2	1	1	3	2.25
	10^5	4	3	5	5	4	3	3	1	3	3.5	3.45
	10^6	5	5	6	6	5	4	5	1	3.5	4	4.45
Decreasing-IW	10^3	7	7	6	7	7	6	7	5	7	7	6.60
	10^4	7	7	7	7	6	6	7	6	7	7	6.70
	10^5	7	6	6	7	6	6	7	6	6	7	6.40
	10^6	1	1.5	5	2	1	1	2.5	6	3.5	4	2.75
Increasing-IW	10^3	2	2	1	2	1	3	2	4	1	1	1.90
	10^4	1	1	1	1	1	2	1	4	4	3	1.90
	10^5	1	1	2	1	2	1.5	3	4	3	3.5	2.20
	10^6	2	1.5	2	3	3	2.5	2.5	5	3.5	4	2.90
Stochastic-IW	10^3	5	5	5	6	5	5	5	2	3	4	4.40
	10^4	4	4	4	4	2	3	4	3	2	3	3.40
	10^5	3	2	3	3	3	1.5	1	3	3	3.5	2.60
	10^6	4	3	3	4	4	2.5	2.5	2	3.5	4	3.25
FIPS	10^3	1	1	2	1	6	1	1	6	2	2	2.30
	10^4	5	6	6	6	7	4	6	7	5.5	3	5.55
	10^5	6	7	7	6	7	5	6	7	7	3.5	6.15
	10^6	7	7	7	7	7	6	7	7	7	4	6.60
HPSOTVAC	10^3	6	6	7	4	2	2	6	7	6	6	5.20
	10^4	6	5	5	5	5	5	5	5	5.5	6	5.25
	10^5	2	5	1	2	1	4	5	5	3	3.5	3.15
	10^6	3	4	1	1	2	5	6	4	3.5	4	3.35
AHPSO	10^3	4	3	4	3	4	7	4	3	5	3	4.00
	10^4	2	2.5	3	2	3	7	3	2	3	3	3.05
	10^5	5	4	4	4	5	7	3	2	3	3.5	4.05
	10^6	6	6	4	5	6	7	2.5	3	3.5	4	4.70

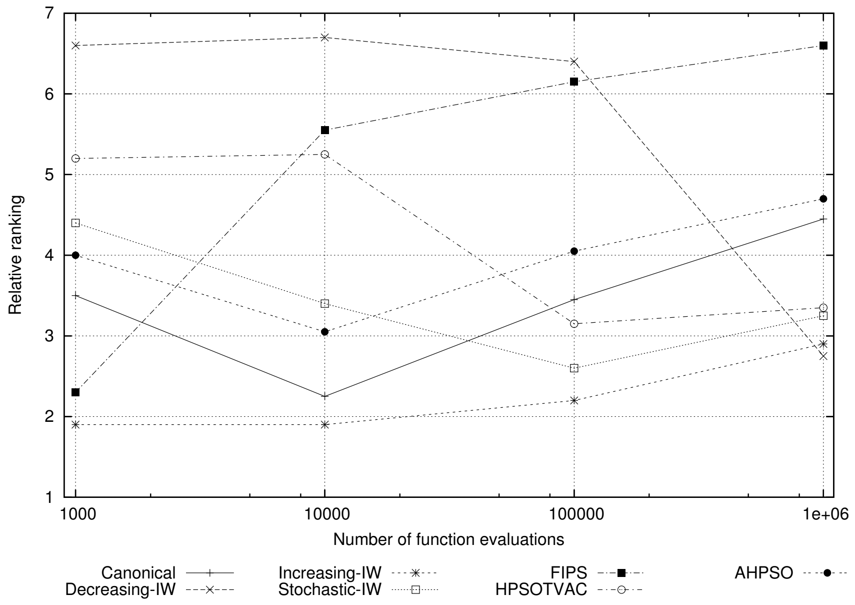


Figure 4.24: Relative average ranking of particle swarm optimizers over time. Fully connected topology case. Average ranking over benchmark functions as a function of the number of function evaluations in a run.

It is clear how the decreasing inertia weight is the worst ranked algorithm most of the time but is first at the very end. It is also interesting to see how the opposite strategy (i.e., increasing the value of the inertia weight over time), is ranked first all the time except at the very end, when they exchange positions.

Something that was not clear on Table 4.2, is how the canonical particle swarm and the stochastic inertia weight particle swarm have a "mirror" behavior, that is, the canonical particle swarm optimizer starts being better ranked than the stochastic inertia weight variant but ends up being worse ranked. In principle, these two variants should behave similarly since the stochastic inertia weight variant is directly inspired on the canonical particle swarm.

A possible explanation to this phenomenon, is that having a fixed inertia weight, sometimes, overconstrains the particle swarm preventing it from continuing exploring the search space.

Table 4.3 shows the particle swarm optimizers' ranking across benchmark functions and function evaluations using a square topology. A less connected topology than the fully connected topology does not permit a direct and instantaneous influence of the very best particle on other particles. The square topology, giving 4 neighbors to every particle, allows the particle swarm to exhibit a more explorative behavior. AHPSO which adapts its topology from a highly connected one to a loosely connected one, is expected to benefit from the early convergent behavior the influence of the very best particle offers.

The promotion of a more explorative behavior in an already-explorative particle swarm optimizer, such as the decreasing inertia weight variant, affects negatively its relative ranking. With the square topology, the decreasing inertia weight particle swarm is no longer the top ranked optimizer in some problems at 1 000 000 function evaluations.

Table 4.3: Ranking of particle swarm optimizers based on the median solution quality achieved after a certain number of function evaluations. Square topology case. Each cell contains the rank obtained by the corresponding optimizer after 1 000, 10 000, 100 000 and 1 000 000 function evaluations (FES).

	Square topology case											
		High dimensional								Low dimensional		
		Multimodal						Unimodal		Multimodal	Unimodal	
	FES	Ackley	Step	Griewank	Rastrigin	Schwefel	Salomon	Sphere	Rosenbrock	Schaffer	Easom	Avg. Rank
Canonical	10^3	5	5	4	4	3	4	4.5	3	3	3	3.85
	10^4	3	3	4	3	3	2	3.5	2	2	3	2.85
	10^5	1	1	2	5	4	2	2	4	3	3.5	2.75
	10^6	2.5	2.5	3	6	5	1.5	3	2	4	4	3.35
Decreasing-IW	10^3	7	7	7	7	7	6	7	4	7	6	6.50
	10^4	7	7	7	7	6	6	7	6	7	7	6.70
	10^5	7	7	7	7	6	6	7	6	7	7	6.70
	10^6	2.5	2.5	3	2	1	3.5	3	4	4	4	2.95
Increasing-IW	10^3	3	3	3	2	1	3	3	2	1	1	2.20
	10^4	4	2	3	2	2	4	5	3	4	3	3.20
	10^5	2	3	2	2	2	2	2	2	3	3.5	2.35
	10^6	2.5	2.5	3	3	3	3.5	3	5	4	4	3.35
Stochastic-IW	10^3	6	5	5	5	4	5	4.5	5	4	4	4.75
	10^4	5	4	5	1	4	3	3.5	4	3	3	3.55
	10^5	3	2	2	4	3	2	2	3	3	3.5	2.75
	10^6	2.5	2.5	3	4	4	1.5	3	3	4	4	3.15
FIPS	10^3	1	1	1	1	6	1	1	6	5	5	2.80
	10^4	1	5	1	5	7	1	2	7	5	3	3.70
	10^5	5	6	6	3	7	5	6	7	6	3.5	5.45
	10^6	6	7	7	5	7	6	7	7	4	4	6.00
HPSOTVAC	10^3	4	5	6	6	2	2	6	7	6	7	5.10
	10^4	6	6	6	6	5	5	6	5	6	6	5.70
	10^5	4	5	4	4	1	4	5	5	3	3.5	3.85
	10^6	5	5	3	1	2	5	6	6	4	4	4.10
AHPSO	10^3	2	2	2	3	5	7	2	1	2	2	2.80
	10^4	2	1	2	1	1	7	1	1	1	3	2.00
	10^5	6	4	5	6	5	7	4	1	3	3.5	4.45
	10^6	7	6	6	7	6	7	3	1	4	4	5.10

A related phenomenon to the one we described above is that the increasing inertia weight variant no longer dominates the others at 1 000 function evaluations as it did in the fully connected topology case. In its stead, FIPS is ranked first in 5 of the 6 high dimensional multimodal functions and in the Sphere function. Contrary to what happened with the fully connected topology, FIPS now maintains this top ranking at 10 000 function evaluations in 4 cases. However, the facilitation of explorative behavior provided by the square topology is not sufficient for FIPS to improve its ranking at 100 000 and 1 000 000 function evaluations.

The canonical particle optimizer seems to take advantage of a different topology and now it is among the top ranked algorithms in four benchmark functions. However, the complete dominance it had in the Rosenbrock function is now transferred to AHPSO. The improvement on the ranking obtained by AHPSO is consequence of the exploitation of a convergent behavior caused by a highly connected topology during the first iterations of the algorithm. This explains why, now, AHPSO is the second best and best algorithm at 1 000 and 10 000 function evaluations respectively, and among the worst afterwards.

Figure 4.25, shows the relative average ranking over time of the compared particle swarm optimizers using a square topology.

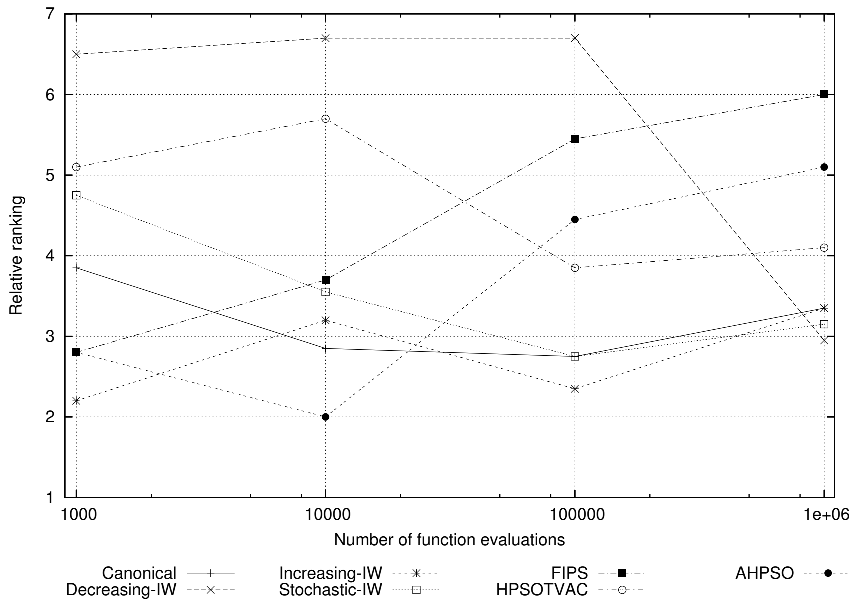


Figure 4.25: Relative average ranking of particle swarm optimizers over time. Square topology case. Average ranking over benchmark functions as a function of the number of function evaluations in a run.

The behavior of the decreasing inertia weight variant is practically the same as the one it had when it used a fully connected topology. During the first 100 000 function evaluations, it is the worst ranked. At the end, it is the top ranked. The early dominance of the increasing inertia weight optimizer is now shared with AHPSO. However, its ranking deteriorates afterwards, and becomes the second worst algorithm just before FIPS.

Table 4.4: Ranking of particle swarm optimizers based on the median solution quality achieved after a certain number of function evaluations. Ring topology case. Each cell contains the rank obtained by the corresponding optimizer after 1 000, 10 000, 100 000 and 1 000 000 function evaluations (FES).

	Ring topology case											
		High dimensional								Low dimensional		
		Multimodal						Unimodal		Multimodal	Unimodal	
	FES	Ackley	Step	Griewank	Rastrigin	Schwefel	Salomon	Sphere	Rosenbrock	Schaffer	Easom	Avg. Rank
Canonical	10^3	5	4.5	4	4	3	4	4	3	3	4	3.85
	10^4	3	4	4	3	4	2	3	3	4	2.5	3.25
	10^5	2	4	1.5	5	5	1	1.5	2	3.5	3.5	2.90
	10^6	3.5	2.5	2.5	6	7	1	3	2	4	4	3.55
Decreasing-IW	10^3	7	7	7	7	7	6	7	4	7	7	6.60
	10^4	7	7	7	7	7	6	7	7	7	7	6.90
	10^5	7	7	7	7	7	6	7	7	7	7	6.90
	10^6	3.5	2.5	5.5	3	3	3	3	4	4	4	3.55
Increasing-IW	10^3	3	2	3	2	1	3	3	2	2	1	2.20
	10^4	4	3	3	2	2	4	4	4	5.5	2.5	3.40
	10^5	3	2.5	1.5	4	2	4	3.5	4	3.5	3.5	3.15
	10^6	3.5	5.5	2.5	4	4	3	3	5	4	4	3.85
Stochastic-IW	10^3	6	6	5	5	4	5	5	6	4	4	5.00
	10^4	5	5	5	4	3	3	5	5	3	2.5	4.05
	10^5	4	2.5	3	3	4	2	3.5	3	3.5	3.5	3.20
	10^6	3.5	2.5	2.5	5	5	3	3	3	4	4	3.55
FIPS	10^3	2	3	2	3	6	1	2	5	5	5	3.40
	10^4	1	2	2	5	6	1	2	2	2	5	2.80
	10^5	1	1	4	2	6	3	1.5	5	3.5	3.5	3.05
	10^6	1	5.5	2.5	2	1	5	3	6	4	4	3.40
HPSOTVAC	10^3	4	4.5	6	6	2	2	6	7	6	6	4.95
	10^4	6	6	6	6	5	5	6	6	5.5	6	5.75
	10^5	5	6	5	1	1	5	6	6	3.5	3.5	4.20
	10^6	6	2.5	5.5	1	2	6	7	7	4	4	4.50
AHPSO	10^3	1	1	1	1	5	7	1	1	1	2	2.10
	10^4	2	1	1	1	1	7	1	1	1	2.5	1.85
	10^5	6	5	6	6	3	7	5	1	3.5	3.5	4.60
	10^6	7	7	7	7	6	7	6	1	4	4	5.60

The canonical particle swarm does not go beyond rank 4 when using the square topology, even after worsening its rank after 100 000 function evaluations. While in the case of the fully connected topology, its rank worsened just after 10 000 function evaluations. The canonical particle swarm seems to take advantage of the explorative behavior encouraged by the square topology.

Table 4.4 shows the particle swarm optimizers' ranking across benchmark functions and function evaluations using the ring topology.

The ring topology, offering only two sources of influence to any particle in a swarm, stimulates a more explorative behavior than the fully connected and square topologies. In our comparison, AHPSO has the advantage of not relying on a fixed topology. It is among the top ranked algorithms when solving either unimodal or low dimensional problems. However, it is ranked among the worst when solving high dimensional multimodal problems at 100 000 and 1 000 000 function evaluations.

The decreasing inertia weight particle swarm is again the worst ranked during the first 100 000 function evaluations and at 1 000 000 it is no longer the top ranked. The relative ranking of FIPS is the one that most dramatically changed. FIPS is among the top ranked optimizers, at least once at some point in time, in all functions except in the Rosenbrock function.

Figure 4.26 shows the particle swarm optimizers' relative average ranking over time when using the ring topology.

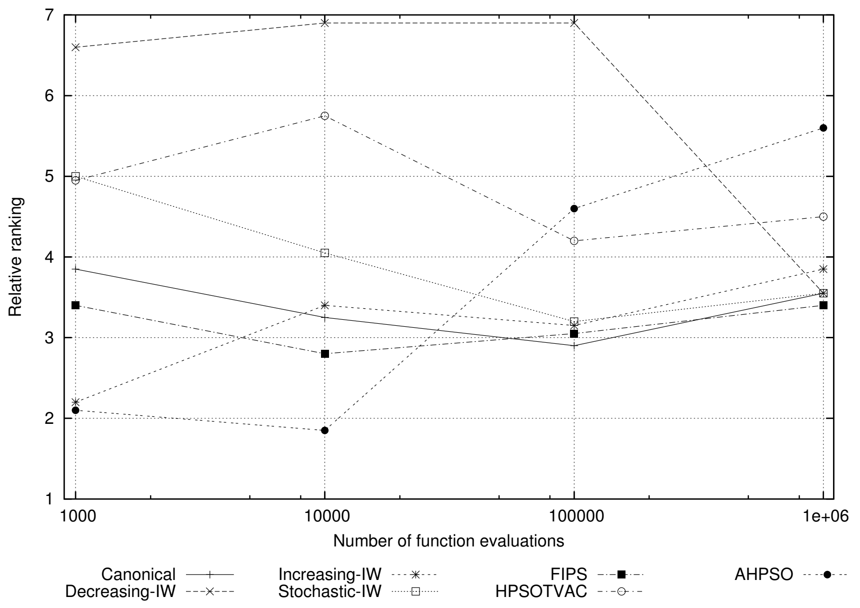


Figure 4.26: Relative average ranking of particle swarm optimizers over time. Ring topology case. Average ranking over benchmark functions as a function of the number of function evaluations in a run.

The top ranked algorithm during the first 10 000 function evaluations is AHPSO. At 100 000 function evaluations, the top ranked algorithm is the canonical particle swarm and at the very end, that is, at 1 000 000 function evaluations, the top ranked algorithm is FIPS. The stochastic inertia weight and the canonical particle swarm do not swap positions this time and they get the same rank at 1 000 000 function evaluations, together with the decreasing inertia weight. HPSOTVAC is the top ranked algorithm when solving the Rastrigin function; however, it is relegated to the worst ranks, on average.

Table 4.5 shows the relative average ranking, of the compared algorithms, obtained with each population topology (sorted from high to low connectivity) and the overall average ranking together with its standard deviation and the corresponding coefficient of variation.

The data in this table reveals the sensitivity that each algorithm has to a topology change. Note that, since AHPSO does not use any of the topologies the other algorithms do, the variation in its relative ranking should be understood as the *indirect* effect of a topology change on the other optimizers in AHPSO's ranking.

The sensitivity to a topology change depends on the budget of number of function evaluations. In almost all algorithms, the effect of a topology change after 1000 function evaluations is the weakest. The exception is AHPSO, in which it is the strongest one.

The most sensitive algorithm to a topology change is FIPS. Considering the topologies' connectivity, the relative average ranking of FIPS decreases as the connectivity decreases, except in the case of 1000 function evaluations, in which the tendency is on the opposite direction. FIPS seems to strongly benefit from the boost in the explorative behavior a loosely connected topology provides. For already-explorative variants (such as HPSOTVAC or the decreasing inertia weight variant), the contrary is true. Their relative average ranking increases as the connectivity decreases. As a consequence, their coefficients of variation increase with the number of function evaluations.

The overall ranking of the increasing inertia weight variant, which almost dominates the others when using a fully connected topology, also increases as the connectivity decreases.

Table 4.5: Relative average ranking of particle swarm optimizers based on the median solution quality achieved after a certain number of function evaluations. Aggregated results. Each cell contains the rank obtained by the corresponding optimizer after 1 000, 10 000, 100 000 and 1 000 000 function evaluations (FES).

Algorithm	FES	Population topology			Average Rank	Standard Deviation	Coefficient of Variation[%]
		Fully Connected	Square	Ring			
Canonical	10^3	3.50	3.85	3.85	3.73	0.20	5.41
	10^4	2.25	2.85	3.25	2.78	0.50	18.08
	10^5	3.45	2.75	2.90	3.03	0.37	12.15
	10^6	4.45	3.35	3.55	3.78	0.59	15.49
Decreasing-IW	10^3	6.60	6.50	6.60	6.57	0.06	0.88
	10^4	6.70	6.70	6.90	6.77	0.12	1.71
	10^5	6.40	6.70	6.90	6.67	0.25	3.77
	10^6	2.75	2.95	3.55	3.08	0.42	13.5
Increasing-IW	10^3	1.90	2.20	2.20	2.10	0.17	8.25
	10^4	1.90	3.20	3.40	2.83	0.81	28.75
	10^5	2.20	2.35	3.15	2.57	0.51	19.90
	10^6	2.90	3.35	3.85	3.37	0.48	14.12
Stochastic-IW	10^3	4.40	4.75	5.00	4.72	0.30	6.39
	10^4	3.40	3.55	4.05	3.67	0.34	9.28
	10^5	2.60	2.75	3.20	2.85	0.31	10.96
	10^6	3.25	3.15	3.55	3.32	0.21	6.28
FIPS	10^3	2.30	2.80	3.40	2.83	0.55	19.44
	10^4	5.55	3.70	2.80	4.02	1.40	34.91
	10^5	6.15	5.45	3.05	4.88	1.63	33.29
	10^6	6.60	6.00	3.40	5.33	1.70	31.89
HPSOTVAC	10^3	5.20	5.10	4.95	5.08	0.13	2.48
	10^4	5.25	5.70	5.75	5.57	0.28	4.95
	10^5	3.15	3.85	4.20	3.73	0.53	14.32
	10^6	3.35	4.10	4.50	3.98	0.58	14.66
AHPSO	10^3	4.00	2.80	2.10	2.97	0.96	32.39
	10^4	3.05	2.00	1.85	2.30	0.65	28.43
	10^5	4.05	4.45	4.60	4.37	0.28	6.51
	10^6	4.70	5.10	5.60	5.13	0.45	8.78

4.3.2 Identification of the Best Performing Particle Swarm Optimizers

Knowing the relative average rank of a particle swarm optimizer is helpful only we if we know which topology gives the best results in terms of actual solution qualities. It might be the case, for example, that the top ranked algorithm using a ring topology yields worse results than the worst ranked algorithm using a fully connected topology. Relative rankings do not say anything about the actual solution quality achieved by the algorithms using one or another topology.

Our goal now is to determine which particle swarm optimizers (from among those we selected, of course) are the best performing and what are the conditions and factors that make them perform at their best. The specific conditions and factors we are talking about are the point in time at which they reach their peak performance, the population topology that makes them get the best results, and the kind of problem they solve best.

Table 4.6 shows various matrices (one matrix per algorithm) that indicate, through an asterisk, on which problem, a given particle swarm optimizer was ranked among the top 3 best optimizers. As a ranking pool, we used all the 18 possible combinations between algorithms and topologies (6 algorithms $\times$ 3 topologies), plus AHPSO which exhibits an adaptive topology mechanism. Sometimes, most commonly on Schaffer and Easom functions, two or more algorithms found solutions with the same quality level. If this quality level was among the top 3 best solution qualities, all these algorithms were considered to be part of the top 3 group. In the rightmost column, the number of times a given algorithm (after a certain number of function evaluations) appears in the top 3 group, is reported.

The matrices in this table show that some algorithms are always (i.e., regardless the number of function evaluations) in the top 3 group for some functions. This happens, for example, in the case of the canonical particle swarm optimizer, the stochastic inertia weight variant, both with a fully connected topology and with AHPSO on the Rosenbrock function. Another example is the increasing inertia weight particle swarm with a fully connected topology on the Step, Sphere, and Easom functions.

There are other cases in which an algorithm appears in the top 3 group most of the time. Examples are the canonical particle swarm optimizer with a fully connected topology on the Sphere, Schaffer and Easom functions, or the fully informed particle swarm with a ring topology on the Ackley and Easom functions.

Note that this "dominance" appears only in two of the high dimensional multimodal functions (Ackley and Step), in high dimensional unimodal functions (Sphere and Rosenbrock) and in low dimensional functions (Schaffer and Easom).

The number of times a given algorithm appears in the top 3 group, tends to increase as the number of function evaluations also increases. However, in some cases, specially those in which an algorithm uses a fully connected topology, the opposite is what happens. This is the case of the canonical, the increasing inertia weight, the stochastic inertia weight, the fully informed particle swarm optimizers with the fully connected topology, and the fully informed particle swarm with a square topology and AHPSO.

Table 4.6: Frequency and distribution of appearances in the top 3 group

Algorithm	Topology	FES	Ackley	Step	Griewank	Rastrigin	Schwefel	Salomon	Sphere	Rosenbrock	Schaffer	Easom	Count
Canonical	Fully Connected	10^3	-	-	-	-	-	-	-	*	-	-	1
		10^4	-	*	*	*	-	*	*	*	*	*	8
		10^5	-	-	-	-	-	-	*	*	*	*	4
		10^6	-	-	-	-	-	-	*	*	*	*	4
	Square	10^3	-	-	-	-	-	-	-	-	-	-	0
		10^4	-	-	-	-	-	-	-	-	-	*	1
		10^5	*	*	*	-	-	*	*	-	*	*	7
		10^6	*	*	*	-	-	*	*	-	*	*	7
	Ring	10^3	-	-	-	-	-	-	-	-	-	-	0
		10^4	-	-	-	-	-	-	-	-	-	*	1
		10^5	-	-	*	-	-	*	*	-	*	*	5
		10^6	*	*	*	-	-	*	*	-	*	*	7
Decreasing-IW	Fully Connected	10^3	-	-	-	-	-	-	-	-	-	-	0
		10^4	-	-	-	-	-	-	-	-	-	-	0
		10^5	-	-	-	-	-	-	-	-	-	-	0
		10^6	*	*	-	-	-	*	*	-	*	*	6
	Square	10^3	-	-	-	-	-	-	-	-	-	-	0
		10^4	-	-	-	-	-	-	-	-	-	-	0
		10^5	-	-	-	-	-	-	-	-	-	*	1
		10^6	*	*	*	-	*	*	*	-	*	*	8
	Ring	10^3	-	-	-	-	-	-	-	-	-	-	0
		10^4	-	-	-	-	-	-	-	-	-	-	0
		10^5	-	-	-	-	-	-	-	-	*	*	2
		10^6	*	*	*	-	-	*	*	-	*	*	7
Increasing-IW	Fully Connected	10^3	*	*	*	*	*	-	*	-	*	*	8
		10^4	*	*	*	*	*	*	*	-	-	*	8
		10^5	*	*	-	-	-	*	*	-	*	*	6
		10^6	-	*	-	-	-	-	*	-	*	*	4
	Square	10^3	-	-	-	-	*	-	-	-	*	*	3
		10^4	-	-	-	-	-	-	-	-	-	*	1
		10^5	-	-	*	*	-	*	*	-	*	*	6
		10^6	*	*	*	-	-	*	*	-	*	*	7

Table 4.6 – continued from previous page

Algorithm	Topology	FES	Ackley	Step	Griewank	Rastrigin	Schwefel	Salomon	Sphere	Rosenbrock	Schaffer	Easom	Count
	Ring	10^3	-	-	-	-	*	-	-	-	-	-	1
		10^4	-	-	-	-	-	-	-	-	-	*	1
		10^5	-	-	*	-	-	-	*	-	*	*	4
		10^6	*	*	*	-	-	*	*	-	*	*	7
Stochastic-IW	Fully Connected	10^3	-	-	-	-	-	-	-	*	-	-	1
		10^4	-	*	-	-	*	*	-	*	*	*	6
		10^5	-	*	-	-	-	*	*	*	*	*	6
		10^6	-	*	-	-	-	-	*	*	*	*	5
	Square	10^3	-	-	-	-	-	-	-	-	-	-	0
		10^4	-	-	-	-	-	-	-	-	-	*	1
		10^5	-	-	*	*	-	*	*	-	*	*	6
		10^6	*	*	*	-	-	*	*	-	*	*	7
	Ring	10^3	-	-	-	-	-	-	-	-	-	-	0
		10^4	-	-	-	-	-	-	-	-	-	*	1
		10^5	-	-	*	-	-	*	*	-	*	*	5
		10^6	*	*	*	-	-	*	*	-	*	*	7
FIPS	Fully Connected	10^3	*	*	*	*	-	*	*	-	*	*	8
		10^4	-	-	-	-	-	-	-	-	-	*	1
		10^5	-	-	-	-	-	-	-	-	-	*	1
		10^6	-	-	-	-	-	-	-	-	*	*	2
	Square	10^3	*	*	*	*	-	*	*	-	-	-	6
		10^4	*	-	*	-	-	-	-	-	-	*	3
		10^5	-	-	-	*	-	-	-	-	*	*	3
		10^6	-	-	-	-	-	-	-	-	*	*	2
	Ring	10^3	-	-	-	-	-	-	-	-	-	-	0
		10^4	*	-	-	-	-	-	-	-	-	*	2
		10^5	*	-	*	-	-	*	*	-	*	*	6
		10^6	*	*	*	-	*	*	*	-	*	*	8
HPSOTVAC	Fully Connected	10^3	-	-	-	-	-	*	-	-	-	-	1
		10^4	-	-	-	-	-	-	-	-	-	*	1
		10^5	-	-	-	-	*	-	-	-	*	*	3
		10^6	-	*	*	*	-	-	-	-	*	*	5

Table 4.6 – continued from previous page

Algorithm	Topology	FES	Ackley	Step	Griewank	Rastrigin	Schwefel	Salomon	Sphere	Rosenbrock	Schaffer	Easom	Count
	Square	10^3	-	-	-	-	-	-	-	-	-	-	0
		10^4	-	-	-	-	-	-	-	-	-	*	1
		10^5	-	-	-	-	*	-	-	-	*	*	3
		10^6	-	*	*	*	-	-	*	-	*	*	6
	Ring	10^3	-	-	-	-	-	-	-	-	-	-	0
		10^4	-	-	-	-	-	-	-	-	-	-	0
		10^5	-	-	-	-	*	-	-	-	*	*	3
		10^6	-	*	*	*	*	-	-	-	*	*	6
AHPSO	Adaptive	10^3	-	-	-	-	-	-	-	*	-	-	1
		10^4	-	*	-	*	*	-	*	*	*	*	7
		10^5	-	-	-	-	-	-	*	*	*	*	4
		10^6	-	-	-	-	-	-	*	*	*	*	4

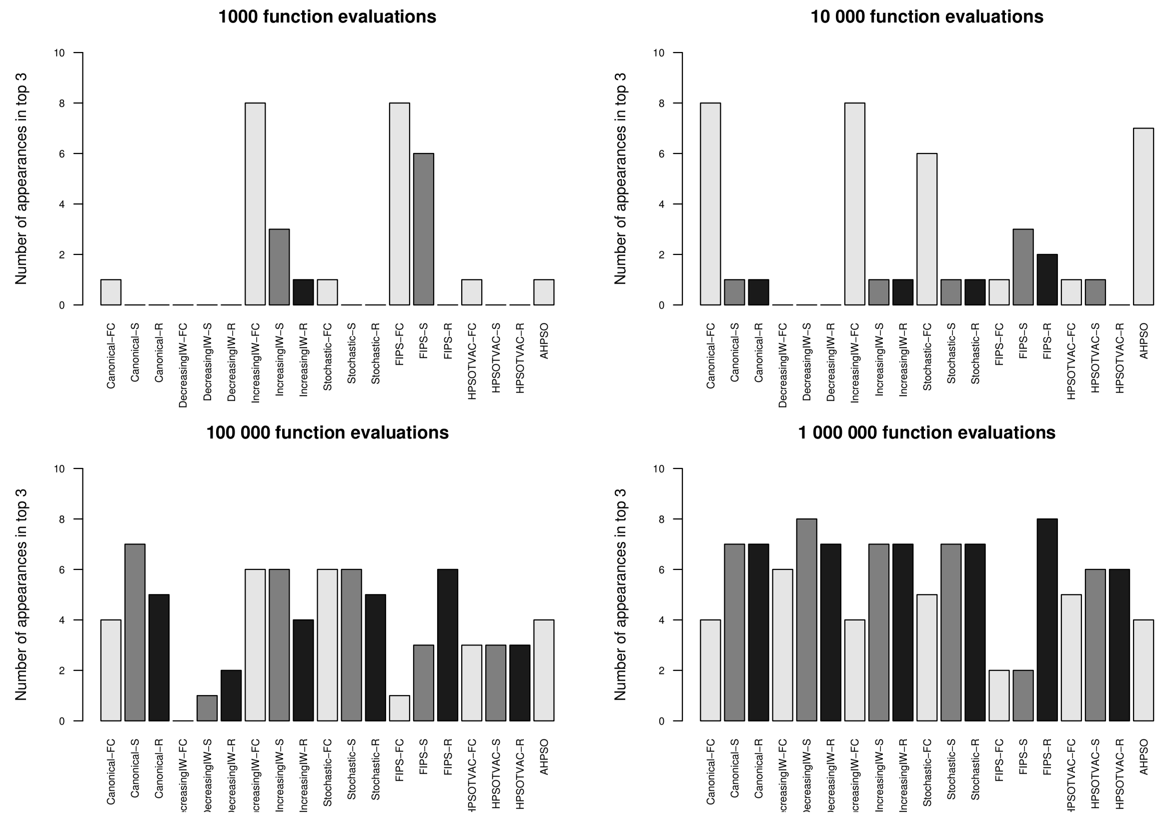


Figure 4.27: Particle swarm optimizers' frequency of appearance in the top 3 group per number of function evaluations. Each figure shows the frequency with which each algorithm appears in the top 3 group after some specific number of function evaluations (specified on the top part of each figure). Shade differences represent different population topologies.

Figure 4.27 shows the frequency with which each algorithm appears in the top 3 group per number of function evaluations. Shade differences represent different population topologies.

On the top-left figure, it can be seen how after 1000 function evaluations, the increasing inertia weight and the fully informed particle swarm optimizers are in the top 3 group 8 times (out of 10 possible). These two same algorithms, but using a square topology, are the other algorithms with more appearances in the top 3 group. The only algorithm using a ring topology that appears at least once, is the increasing inertia weight algorithm.

On the top-right figure, the two algorithms with the highest frequency of appearance (8 times) are the increasing inertia weight and the canonical particle swarm optimizers. They are followed by AHPSO with 7 times and by the stochastic inertia weight variant with 6 times. Except for AHPSO, all these algorithms used a fully connected topology. FIPS with a square topology and with a ring topology appeared three and two times, respectively. At 10 000 function evaluations, FIPS with a fully connected topology is no longer among the best algorithms. The frequency of appearance of FIPS with a square topology dropped from 6 at 1000 function evaluations to just 3 at 10 000.

Its clear that for budgets of relatively few function evaluations, algorithms with a strong convergent behavior obtain the best results. The best algorithms up to 10 000 function evaluations are those that at least start (in case they adapt it, like AHPSO) with a strongly connected topology, and reduce the particles' previous velocity effect with a small inertia weight.

On the bottom-left figure, almost all algorithms appear at least once in the top 3 group. The algorithm with the highest frequency is the canonical particle swarm optimizer with a square topology. The increasing inertia weight with a fully connected or square topology appears 6 times, as well as FIPS with a ring topology and the stochastic inertia weight with a fully connected or square topology. FIPS is the algorithm that uses a ring topology with the highest frequency. There is no clear dominance of any topology. However, algorithms using the ring topology increased substantially the number of times they appeared in the top 3 group.

On the bottom-right figure, all algorithms appear, at least, a couple of times in the top 3 group. The two functions in which all algorithms appear in the top 3 group are Schaffer and Easom (see Table 4.6). In fact, these produce a ceiling effect [8] at a frequency count of two. In any case, the relative frequency difference is still meaningful. The two algorithms with the highest frequency are the decreasing inertia weight using a square topology, and FIPS with the ring topology.

From the two bottom figures, it is clear that for medium-long and long budgets, algorithms with medianly or loosely connected topologies yield the best results. In the bottom-left figure, no algorithm with a fully connected topology have a higher frequency than an algorithm with a square topology. In the bottom-right figure, algorithm with the square and ring topologies share out the highest frequencies.

In terms of accumulated frequencies across number of function evaluations, the two algorithms with the highest ones are the increasing inertia weight particle swarm optimizer with a fully connected topology (26 times out of 40 possible) and the stochastic inertia weight variant with a fully connected topology (18 times).

4.4 Different Inertia Weight Schedules in the Time-Decreasing Inertia Weight Particle Swarm Optimizer

The analysis carried out in the last sections, on the performance of different particle swarm optimizers, revealed that, in most cases, the time-decreasing inertia weight variant is capable of achieving high quality solutions although it is normally among the slowest variants. Another interesting fact is that the time-increasing inertia weight variant is, in many cases, among the fastest variants. Its problem is that it has a strong stagnating behavior. Combining the fast convergence properties of the time-increasing inertia weight variant with the reliability and the high-quality reaching properties of the decreasing inertia weight variant would certainly produce a high performance particle swarm optimizer (compared with those optimizers included in our study, of course).

The time-decreasing inertia weight variant was designed with an adaptive diversification-intensification behavior in mind. A decreasing inertia weight would give the particle swarm the opportunity to roughly explore the search space at the beginning of the run and the ability to move slowly at the end, in the most promising region. However, the inertia weight schedule was originally proposed to decrease its value from a maximum to a minimum over the whole optimization process. We hypothesized that speeding up the inertia weight schedules would force the decreasing inertia weight particle swarm optimizer to converge faster, while retaining its high-quality reaching properties.

In this section, we present the effects of different inertia weight schedules on the performance of the time-decreasing inertia weight particle swarm optimizer. As we did earlier, we only discuss the run-length and solution-quality distributions obtained in some of the benchmark problems. The remaining results are available online at http://iridia.ulb.ac.be/supp/IridiaSupp2006-005/index.html.

The inertia weight schedule we used in the experiments, whose results we present in this section, is based on Equation 2.7. We introduce a slight modification of it, so that whenever the inertia weight reaches its minimum value, it remains there. The modified equation is

$$w(t) = \begin{cases} \frac{t_{max}-t}{t_{max}} \cdot (iw_{max} - iw_{min}) + iw_{max} & \text{if } t \leq t_{max} \\ iw_{min} & \text{otherwise} \end{cases} , \tag{4.11}$$

where t_{max} marks the time at which $w(t) = iw_{min}$, iw_{max} and iw_{min} are the maximum and minimum values the inertia weight can take, respectively.

Besides presenting the run-length and solution-quality distributions obtained for each inertia weight schedule, we also present the development of the swarm's instantaneous speed over time and the development of the average distance between particles and the swarm's centroid. These two measures give an idea of the exploring/exploiting behavior of the algorithms.

The swarm's instantaneous speed at time step t is computed using

$$ss_t = \frac{1}{n}\sum_{i=1}^{n} |\vec{v}_i^t| , \tag{4.12}$$

where n is the number of particles in the swarm and $\vec{v}_i^t$ is the velocity vector of the ith particle at time step t.

To monitor the diversity of the population at a given time step t, we use the average distance between particles and the swarm's centroid. It is computed using

$$d_t = \frac{1}{n} \sum_{j=1}^{n} \left| \vec{x}_j^t - \frac{1}{n} \sum_{i=1}^{n} \vec{x}_i^t \right| , \tag{4.13}$$

where n is the number of particles in the swarm and $\vec{x}_i^t$ is the position vector of the ith particle at time step t.

A fast-moving disperse swarm is an indicator of an explorative behavior in a particle swarm optimizer. A slowly-moving contracted particle swarm is an indicator of a particle swarm optimization algorithm in exploitation mode. During a run, we monitor these two measures to understand the effect of the decreasing inertia weight, and its different schedules, on the exploration/exploitation behavior of the algorithm.

4.4.1 Effects of Different Inertia Weight Schedules on the Easom Function

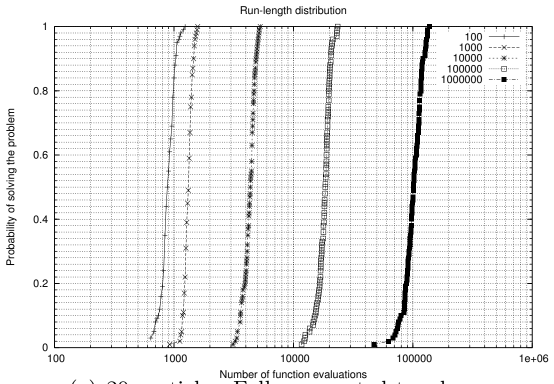
Figure 4.28 shows the empirical run-length distributions of the decreasing inertia weight particle swarm optimizer with different inertia weight schedules. The demanded solution quality was of 0.0001% away from the global optimum.

Irrespective of the number of particles and the population topology, the effect of different inertia weight schedules is the same. The more aggressive a schedule is, the higher is the speed up in the optimizer's performance. No degradation in the probability of finding the required solution can be perceived. The difference in the speed up (measured as the distance between distributions along the x-axis) between each schedule, is influenced by the number of particles and the population topology. With 20 particles, the difference is higher than with 60 particles, and it is also higher using the fully connected topology than using the ring topology. The speed up, in most of the cases, is substantial.

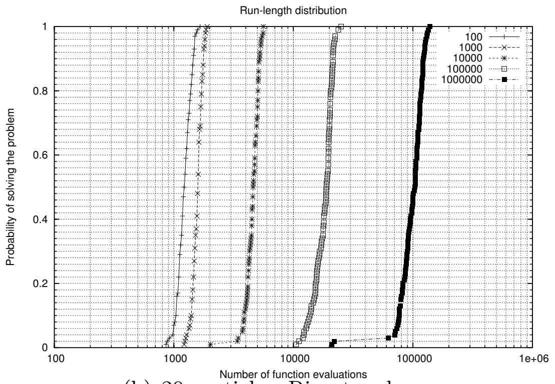
Figures 4.29 and 4.30 show the particles' average distance to the swarm's centroid and their average instantaneous speed, respectively. In all cases, faster inertia weight schedules provoke faster reductions of the swarm's average speed and faster spatial contractions of the particle swarm. The effects of an increment in the number of particles are similar to those of a decrement in the connectivity of the population topology: both, the speed and the distance towards the swarm's centroid, do not decrease at the same rate. With more particles or loosely connected topologies, the decrements in the particles' speed and the shrinkage of the particles' distance to the swarm's centroid are slower.

Figure 4.31 shows the solution-quality distributions obtained by the decreasing inertia weight particle swarm optimizer with different inertia weight schedules after 1000 function evaluations. Fast inertia weight schedules provoke an increment in the probability of finding high quality solutions after 1000 function evaluations. An increment in the number of particles reduces the probability significantly. A change in the population topology towards loosely connected topologies, also provokes a decrement in the probability of finding good solutions, although not as significantly as an increment in the number of particles.

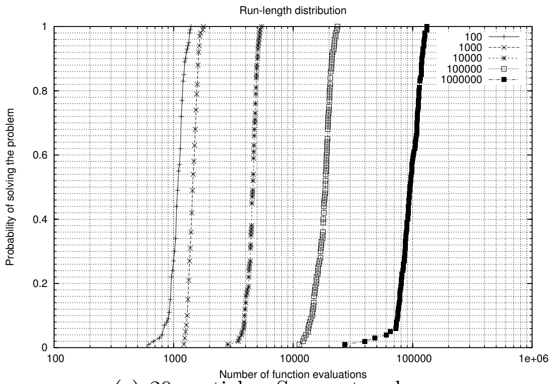
Figure 4.32 shows the curves of the median solution quality development over time for different inertia weight schedules. The fastest configuration is the one using 20 particles and a fully connected topology.



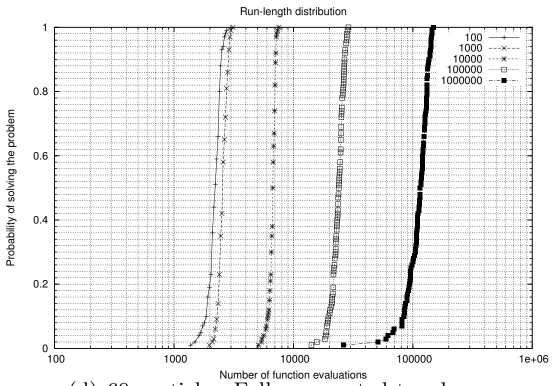
(a) 20 particles, Fully connected topology



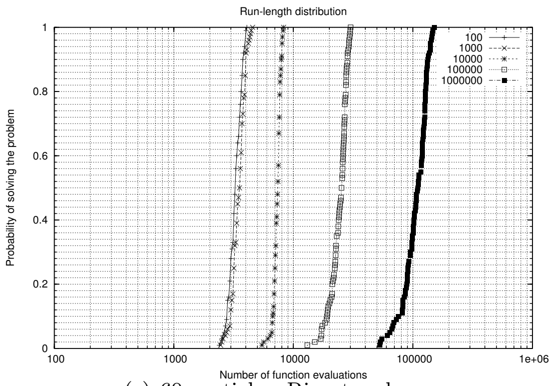
(b) 20 particles, Ring topology



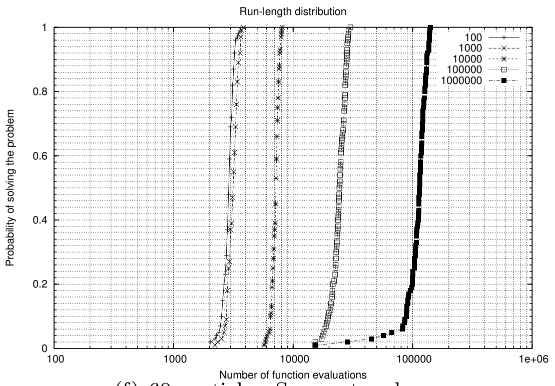
(c) 20 particles, Square topology



(d) 60 particles, Fully connected topology

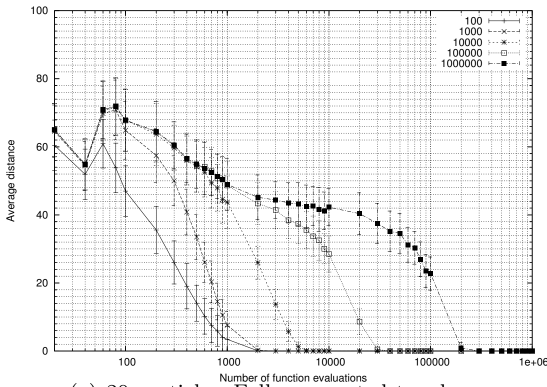


(e) 60 particles, Ring topology

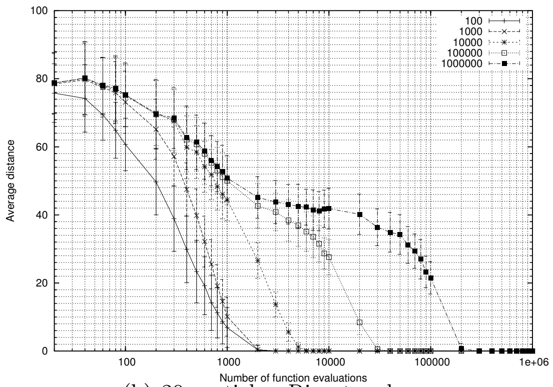


(f) 60 particles, Square topology

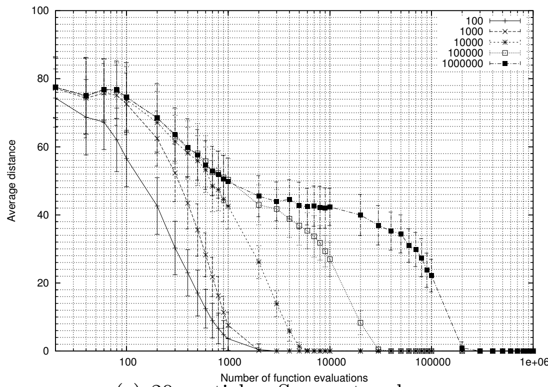
Figure 4.28: Run-length distributions obtained with different inertia weight schedules on the Easom function. The solution quality demanded is of 0.0001% away from the global optimum. Figures (a), (b), and (c) show the run-length distributions obtained with 20 particles. Figures (d), (e), and (f) show the run-length distributions obtained with 60 particles. Columns show the effect of using different population topologies.



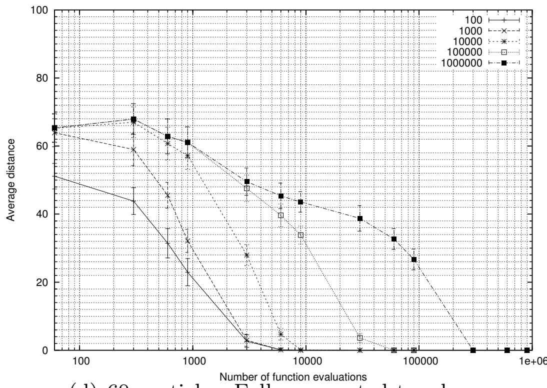
(a) 20 particles, Fully connected topology



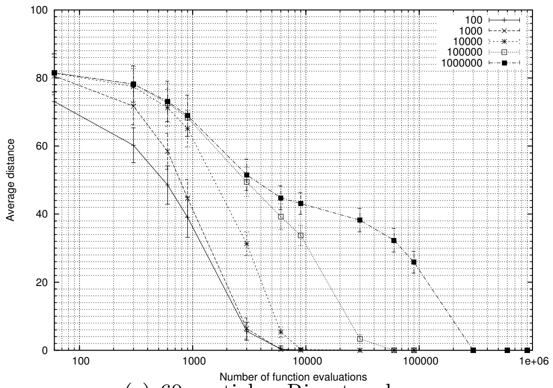
(b) 20 particles, Ring topology



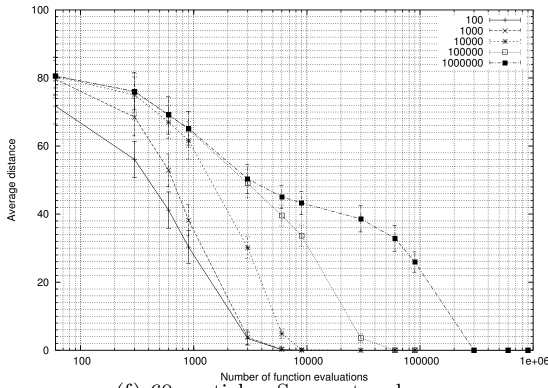
(c) 20 particles, Square topology



(d) 60 particles, Fully connected topology

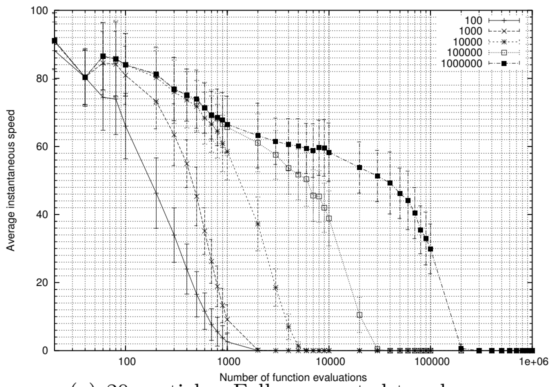


(e) 60 particles, Ring topology

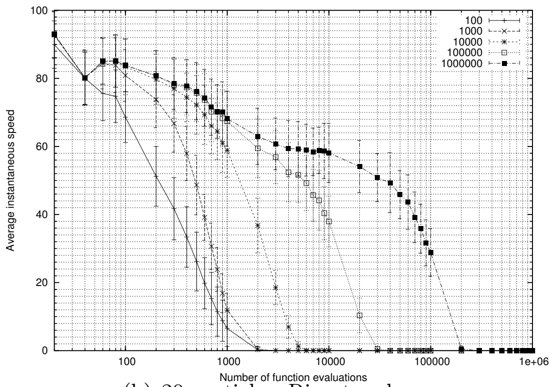


(f) 60 particles, Square topology

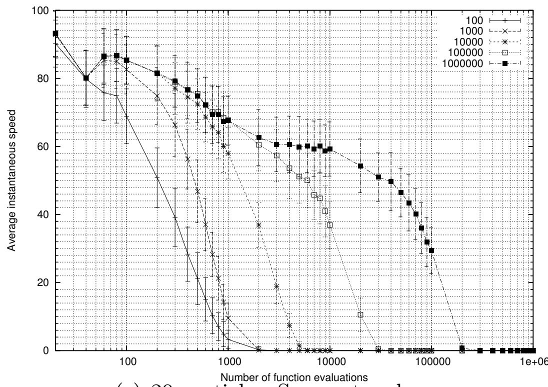
Figure 4.29: Development of the particles' average distance to the swarm's centroid for different inertia weight schedules on the Easom function. Figures (a), (b), and (c) show the results with 20 particles. Figures (d), (e), and (f) show the results with 60 particles. Columns show the effect of using different population topologies.



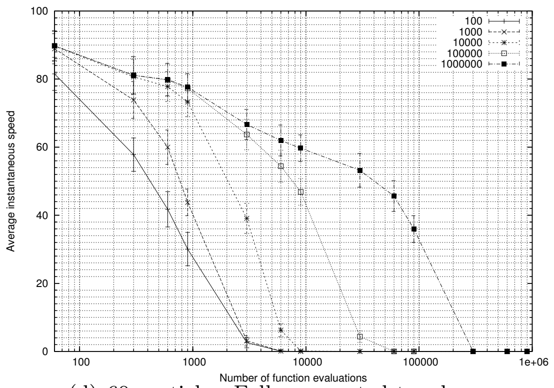
(a) 20 particles, Fully connected topology



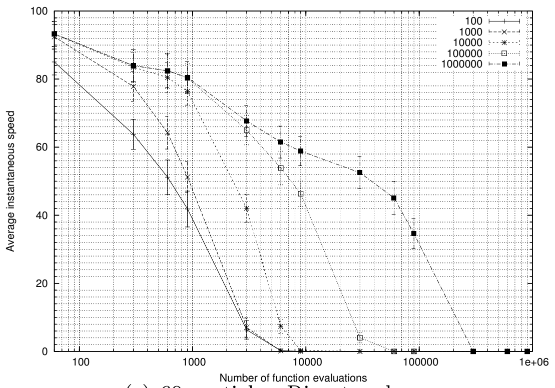
(b) 20 particles, Ring topology



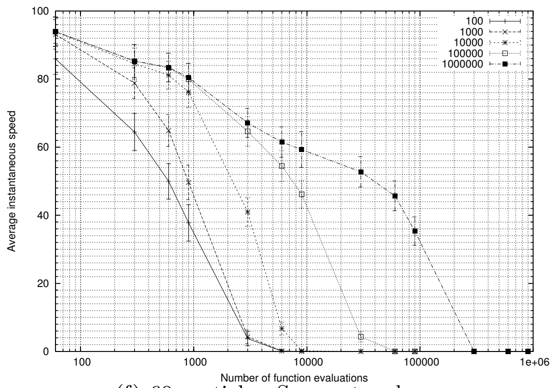
(c) 20 particles, Square topology



(d) 60 particles, Fully connected topology

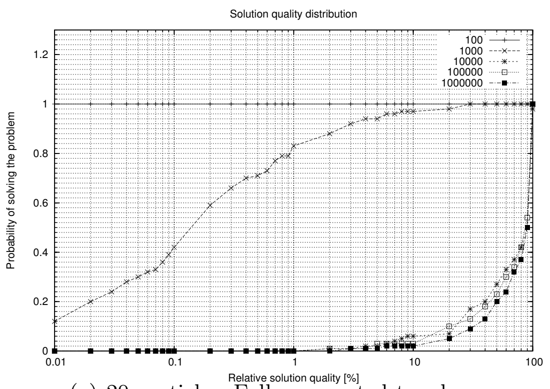


(e) 60 particles, Ring topology

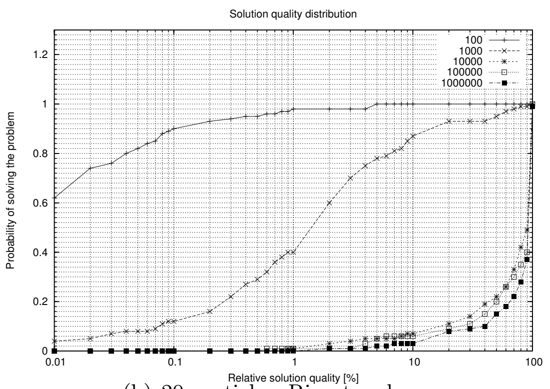


(f) 60 particles, Square topology

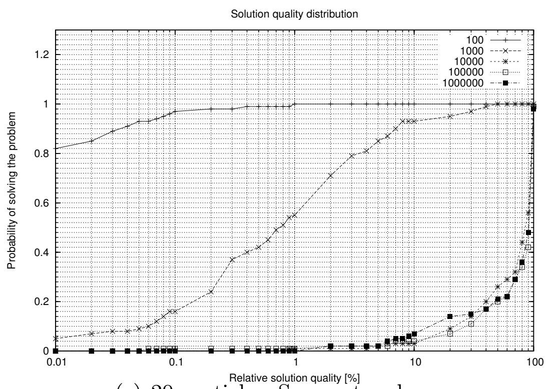
Figure 4.30: Development of the particles' average instantaneous speed for different inertia weight schedules on the Easom function. Figures (a), (b), and (c) show the results obtained with 20 particles. Figures (d), (e), and (f) show the results obtained with 60 particles. Columns show the effect of using different population topologies.



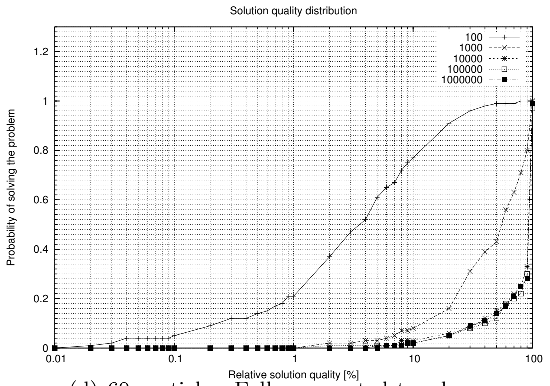
(a) 20 particles, Fully connected topology



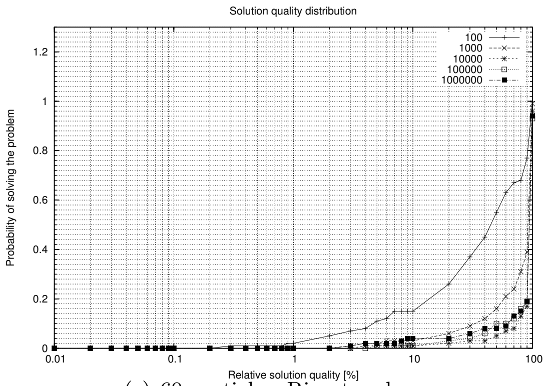
(b) 20 particles, Ring topology



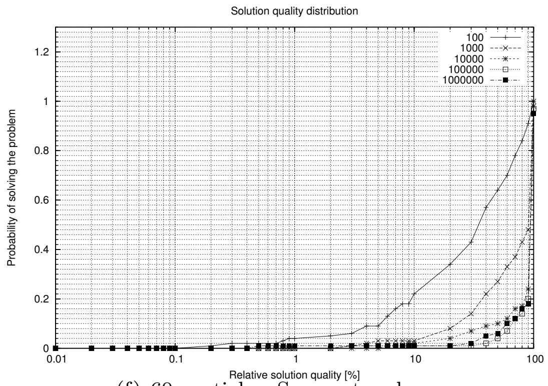
(c) 20 particles, Square topology



(d) 60 particles, Fully connected topology

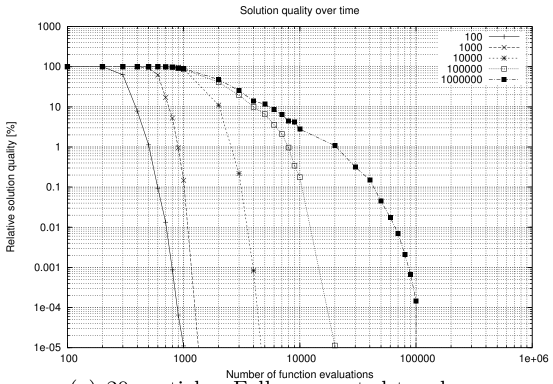


(e) 60 particles, Ring topology

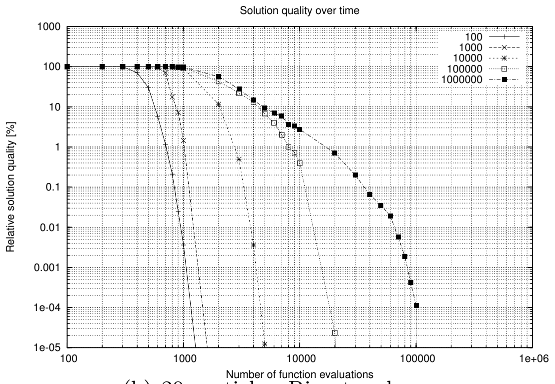


(f) 60 particles, Square topology

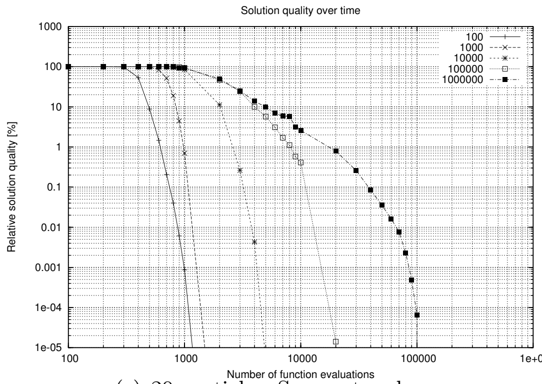
Figure 4.31: Solution-quality distributions for different inertia weight schedules on the Easom function. These results were obtained after 1000 objective function evaluations. Figures (a), (b), and (c) show the solution-quality distributions obtained with 20 particles. Figures (d), (e), and (f) show the solution-quality distributions obtained with 60 particles. Columns show the effect of using different population topologies.



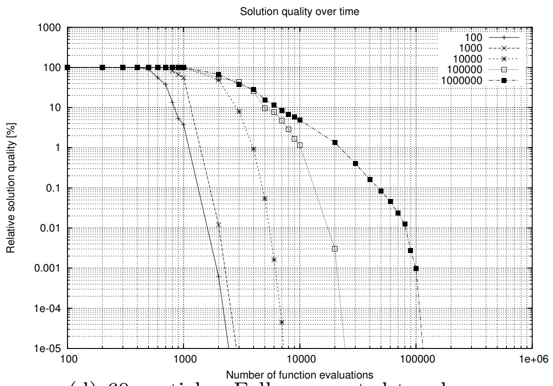
(a) 20 particles, Fully connected topology



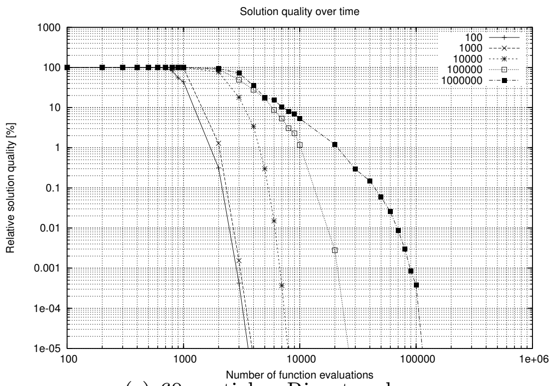
(b) 20 particles, Ring topology



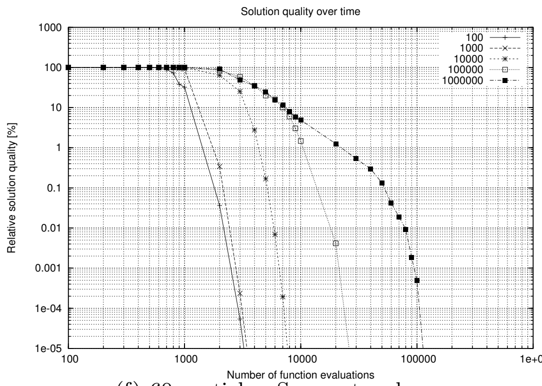
(c) 20 particles, Square topology



(d) 60 particles, Fully connected topology



(e) 60 particles, Ring topology



(f) 60 particles, Square topology

Figure 4.32: Solution quality development over time for different inertia weight schedules on the Easom function. Figures (a), (b), and (c) show the solution quality development over time that is obtained with 20 particles. Figures (d), (e), and (f) show the solution quality development over time that is obtained obtained with 60 particles. Columns show the effect of using different population topologies. These results are based on medians.

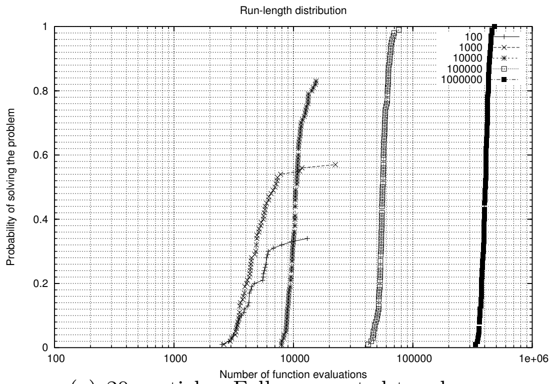
4.4.2 Effects of Different Inertia Weight Schedules on the Rastrigin Function

Figure 4.33 shows the run-length distributions obtained by the decreasing inertia weight particle swarm optimizer with different inertia weight schedules at a solution quality of 20.0% away from the global optimum. Similarly to the case with the Easom function, faster schedules make the algorithm behave more greedily. In some cases, this greediness results in a substantial speed up. For example, when the schedule is decreased from 1 000 000 to 100 000 function evaluations. However, as more aggressive the schedules are, the stronger the stagnating behavior the algorithm exhibits. The severity of the stagnating behavior is alleviated by both, an increase in the number of particles, and a loosely connected topology.

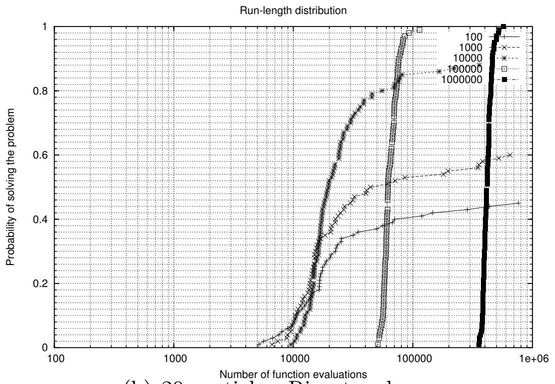
Figures 4.34 and 4.35 show the particles' average distance to the swarm's centroid and their average instantaneous speed, respectively. The rate at which the particles' average speed and their distance to the swarm's centroid decrease, is faster when using a faster schedule. Comparing Figures 4.33(a), 4.33(b), and 4.33(c), for example, two things are clear. First, when using either a ring or square topology, the algorithm does not collapse as it does when using the fully connected topology. Second, at any point in time, the dispersion of the swarm increases as the connectivity of the topology increases. In terms of speed, the difference between topologies is not as high as with the dispersion, and the speed does not drop to zero when the algorithm uses the ring and square topologies as it does when using the fully connected topology. The effect of an increase in the number of particles is not as dramatic as is the effect of a different population topology.

Figure 4.36 shows the solution-quality distributions for different inertia weight schedules obtained after 1 000 000 function evaluations. Irrespective of the number of particles and the population topology, the highest probability of finding good quality solutions is exhibited by the configuration with the slowest schedule. The worst is exhibited by the configuration with the fastest one.

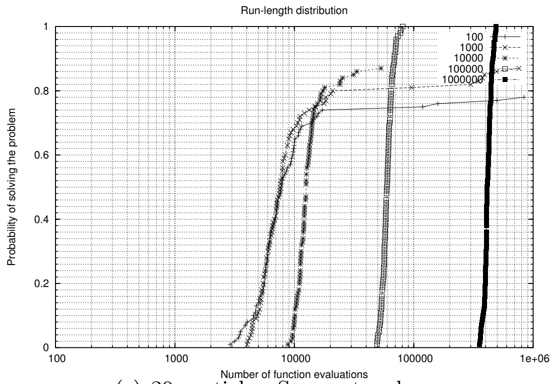
Figure 4.37 shows the solution quality development over time curves for different inertia weight schedules. Even though slow schedules perform very poorly during the first phase in the optimization process, they are the ones that are capable of finding the best quality solutions. The graphs shown in Figure 4.37 show clearly the trade-off between fast convergence and high quality reaching capabilities.



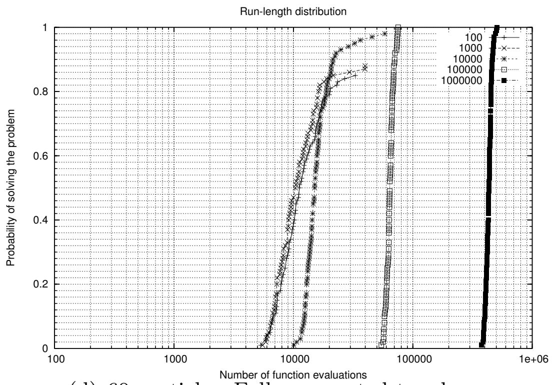
(a) 20 particles, Fully connected topology



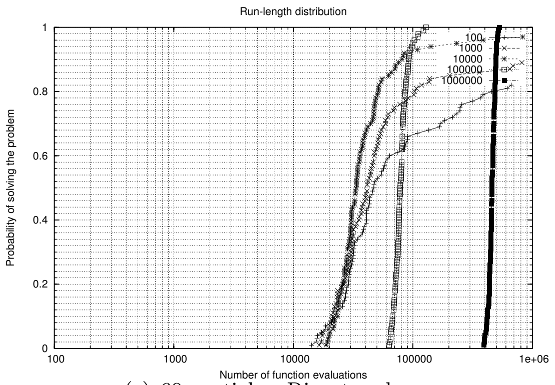
(b) 20 particles, Ring topology



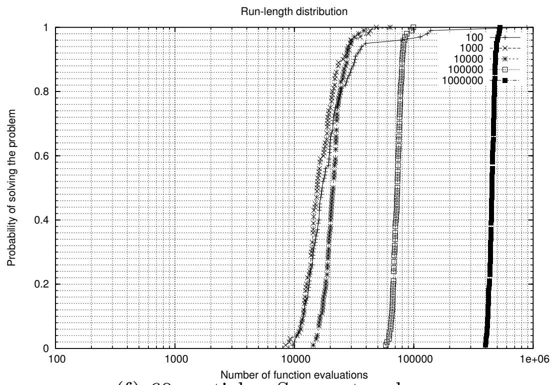
(c) 20 particles, Square topology



(d) 60 particles, Fully connected topology

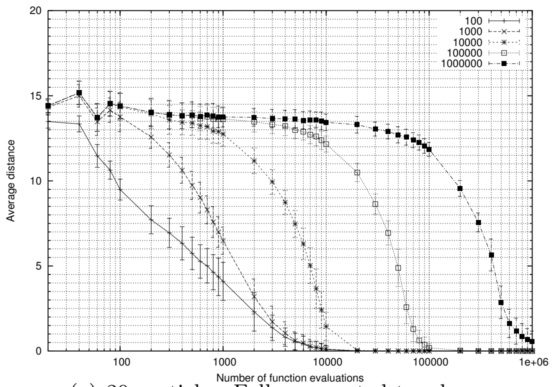


(e) 60 particles, Ring topology

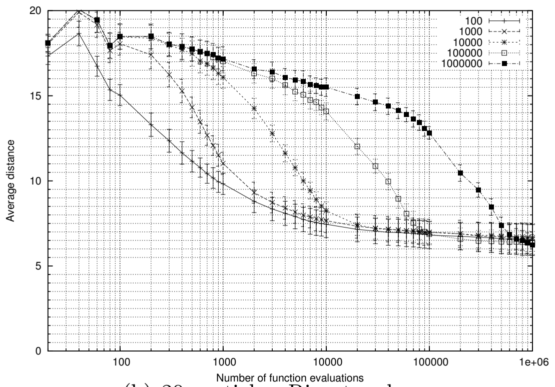


(f) 60 particles, Square topology

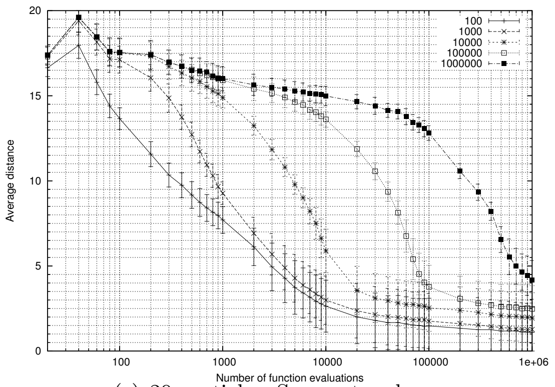
Figure 4.33: Run-length distributions obtained with different inertia weight schedules on the Rastrigin function. The solution quality demanded is of 20.0% away from the global optimum. Figures (a), (b), and (c) show the run-length distributions obtained with 20 particles. Figures (d), (e), and (f) show the run-length distributions obtained with 60 particles. Columns show the effect of using different population topologies.



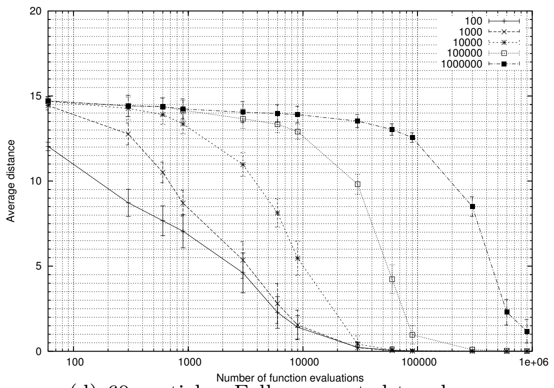
(a) 20 particles, Fully connected topology



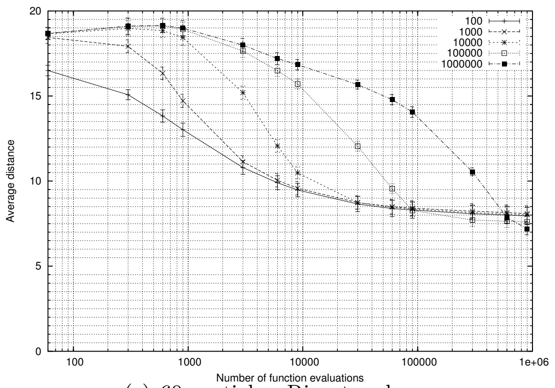
(b) 20 particles, Ring topology



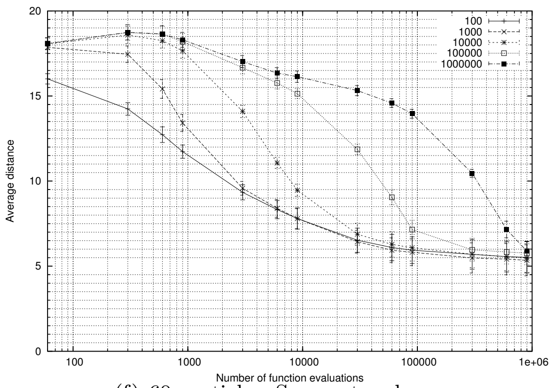
(c) 20 particles, Square topology



(d) 60 particles, Fully connected topology

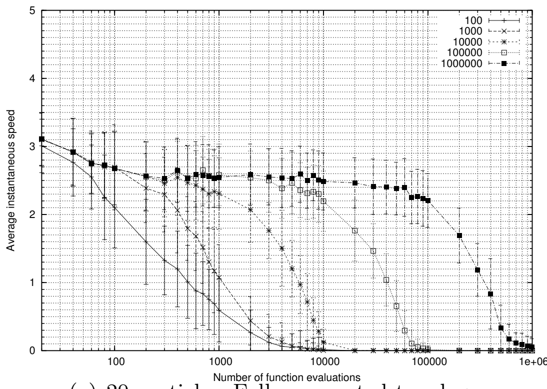


(e) 60 particles, Ring topology

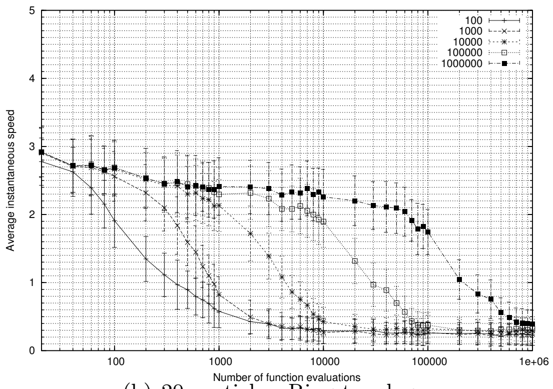


(f) 60 particles, Square topology

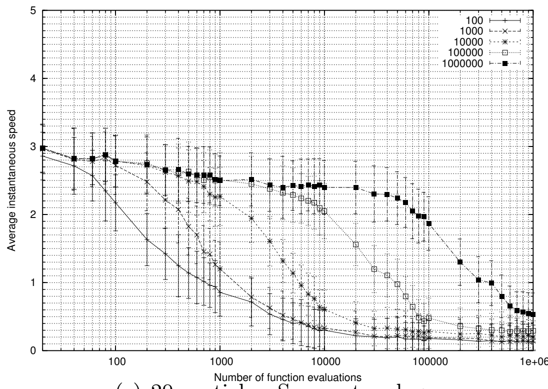
Figure 4.34: Development of the particles' average distance to the swarm's centroid for different inertia weight schedules on the Rastrigin function. Figures (a), (b), and (c) show the results with 20 particles. Figures (d), (e), and (f) show the results with 60 particles. Columns show the effect of using different population topologies.



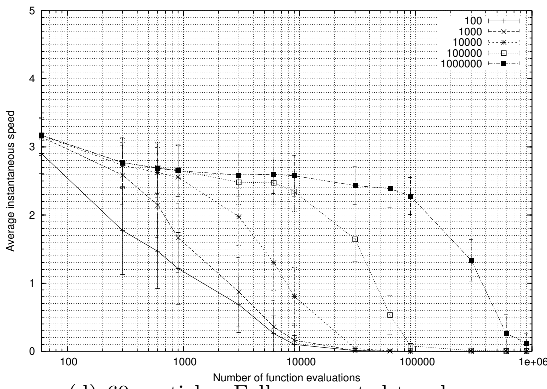
(a) 20 particles, Fully connected topology



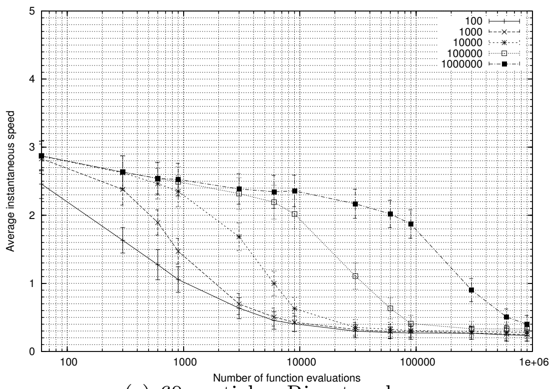
(b) 20 particles, Ring topology



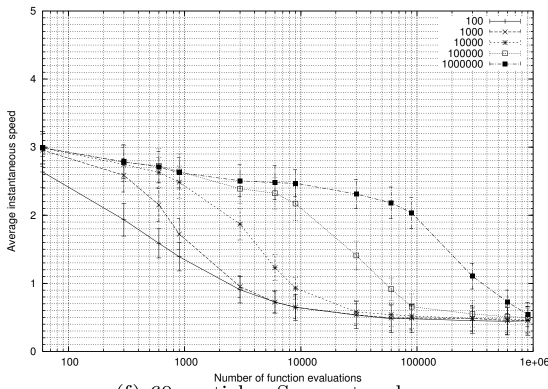
(c) 20 particles, Square topology



(d) 60 particles, Fully connected topology

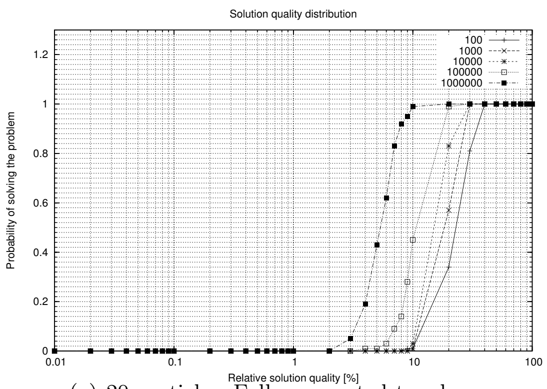


(e) 60 particles, Ring topology

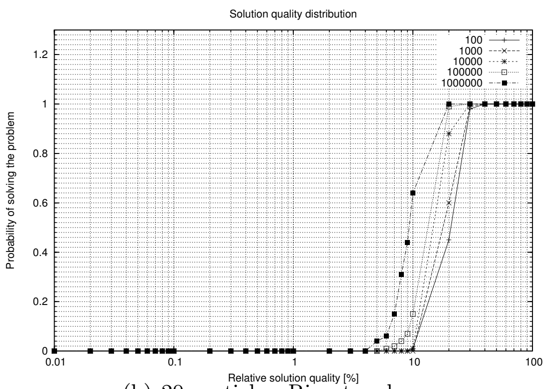


(f) 60 particles, Square topology

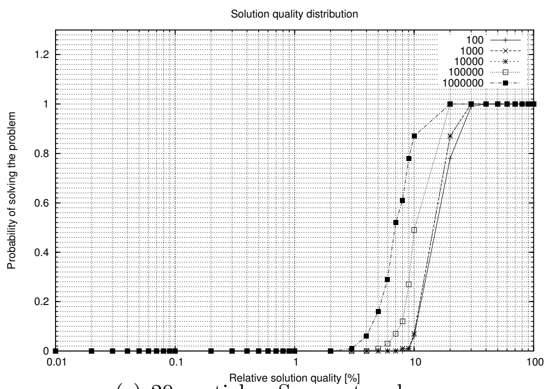
Figure 4.35: Development of the particles' average instantaneous speed for different inertia weight schedules on the Rastrigin function. Figures (a), (b), and (c) show the results obtained with 20 particles. Figures (d), (e), and (f) show the results obtained with 60 particles. Columns show the effect of using different population topologies.



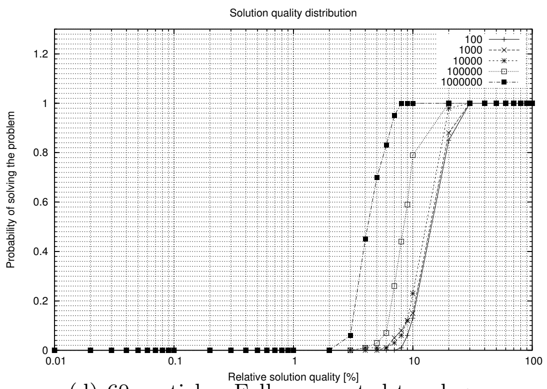
(a) 20 particles, Fully connected topology



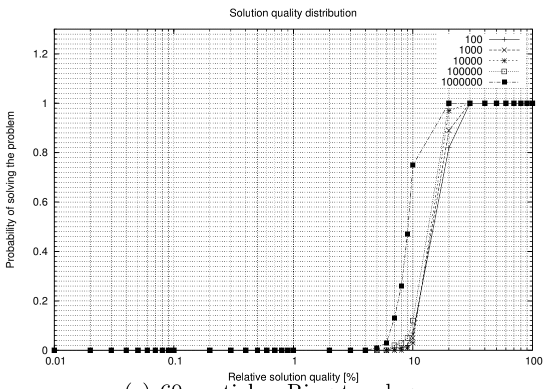
(b) 20 particles, Ring topology



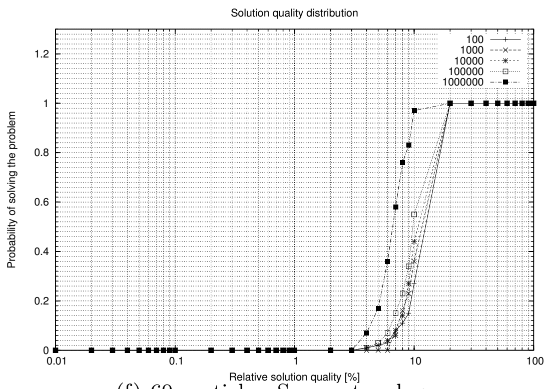
(c) 20 particles, Square topology



(d) 60 particles, Fully connected topology

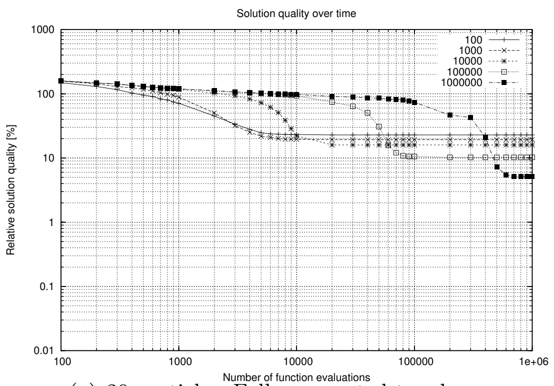


(e) 60 particles, Ring topology

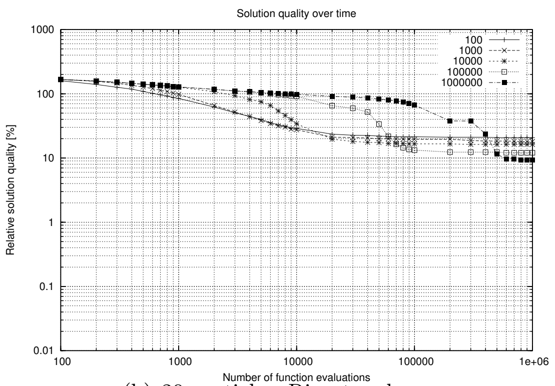


(f) 60 particles, Square topology

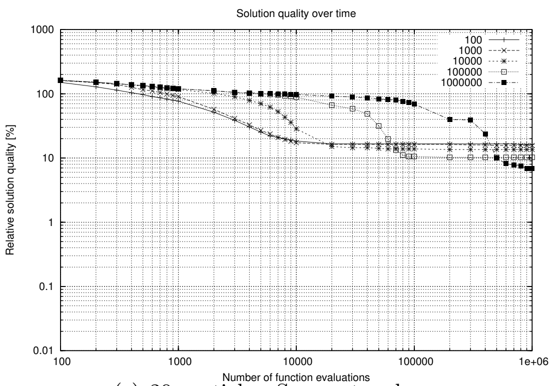
Figure 4.36: Solution-quality distributions for different inertia weight schedules on the Rastrigin function. These results were obtained after 1 000 000 objective function evaluations. Figures (a), (b), and (c) show the solution-quality distributions obtained with 20 particles. Figures (d), (e), and (f) show the solution-quality distributions obtained with 60 particles. Columns show the effect of using different population topologies.



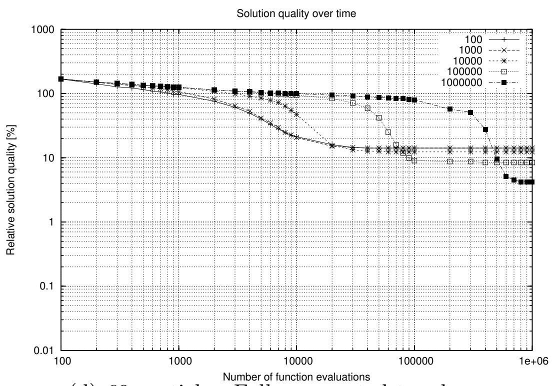
(a) 20 particles, Fully connected topology



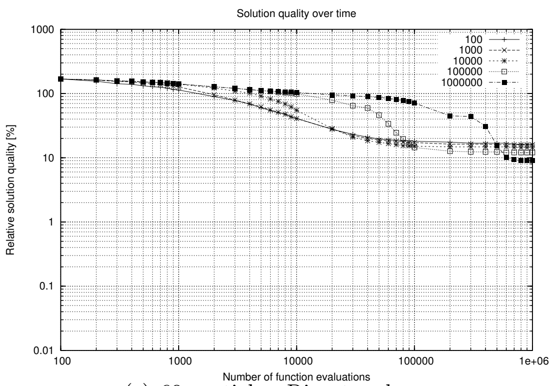
(b) 20 particles, Ring topology



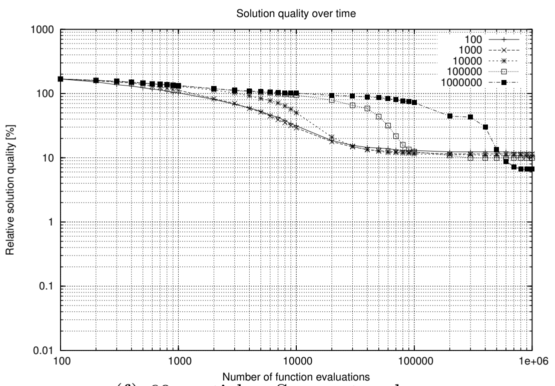
(c) 20 particles, Square topology



(d) 60 particles, Fully connected topology



(e) 60 particles, Ring topology



(f) 60 particles, Square topology

Figure 4.37: Solution quality development over time for different inertia weight schedules on the Rastrigin function. Figures (a), (b), and (c) show the solution quality development over time that is obtained with 20 particles. Figures (d), (e), and (f) show the solution quality development over time that is obtained obtained with 60 particles. Columns show the effect of using different population topologies. These results are based on medians.

4.4.3 Effects of Different Inertia Weight Schedules on the Rosenbrock Function

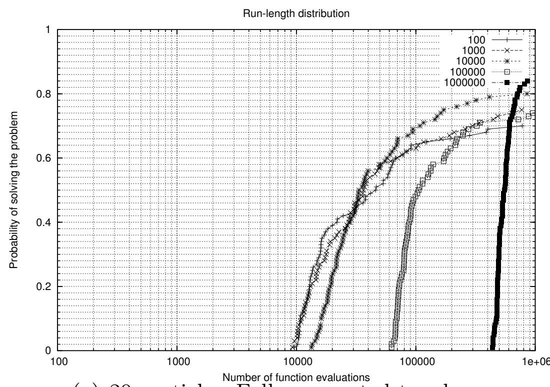
Figure 4.38 shows the run-length distributions obtained by the decreasing inertia weight particle swarm optimizer using different inertia weight schedules. The demanded solution quality was of 10.0% away from the optimum. In the results obtained using the fully connected topology, the most aggressive schedules allow the algorithm to find the desired solution quality level in a faster way. However, in all cases, the algorithm has a strong stagnating behavior, which is not alleviated by an increase in the number of particles. With the ring and square topologies, the algorithm gets better results. There is no clear dominance among fast and not-so-fast schedules. However, the slowest schedule is always dominated.

Figure 4.39 shows the development over time of the particles' average distance to the swarm's centroid. These graphs show how the more connected the population topology, the more contracted the swarm gets. In the case of the ring topology, there is even a significant expansion of the swarm between the first 100 and 1000 function evaluations. The average distance to the swarm's centroid reaches a minimum and then starts to grow again, most notably with the fully connected and square topologies. With the ring topology, the average distance never reaches the zero value. The rate at which the distance decreases depends on the inertia weight schedule. The fastest the schedule, the fastest is the contraction of the swarm. An increase in the number of particles increases the maximum average distance to the swarm's centroid. In other words, bigger swarms are sparser.

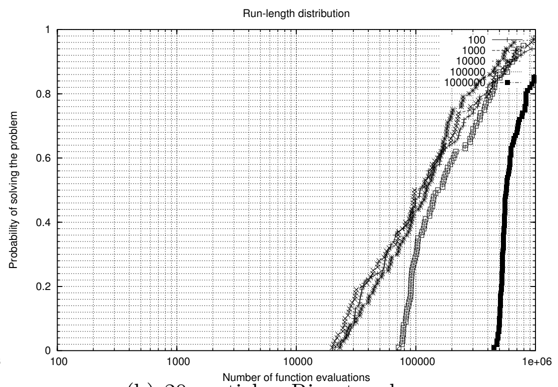
Figure 4.40 shows the average particles' speed development over time. The faster the schedule, the faster the decrement of the speed. In the case of the two fastest schedules, however, the speed decay rate is almost the same after some 1000 function evaluations. The speed also decays up to a minimum and either stays there constant (as in the case of the fully connected topology), or increases (as in the case of the square topology).

Figure 4.41 shows the solution-quality distributions obtained for different inertia weight schedules on the Rosenbrock function. When the algorithm used either a ring or a square topology, the worst results were obtained when the algorithm used the slowest schedule. On the contrary, when the algorithm used the fully connected topology, the best results were obtained when the algorithm used the slowest schedule. Depending on the topology and the number of particles, the best results were obtained when the algorithm used different schedules. For example, with 20 particles and the ring topology the schedule that got the best results was the one that reached the minimum inertia weight after 100 000 function evaluations. With 60 particles and the ring topology the best results were those the algorithm obtained using the inertia weight that reached the minimum inertia weight after 10 000 function evaluations.

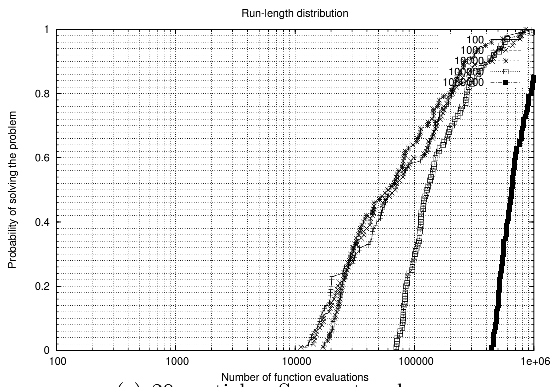
Figure 4.42 shows the graphs of the solution quality development over time. The fastest configuration of the algorithm is the one with 20 particles and the square topology. In this problem, a topology whose connectivity is between a strong one and a loose one gives the best results. Incrementing the number of particles only delays the convergence of the algorithm to high quality solutions, and in this case, the performance obtained with 20 and 60 particles is not so different.



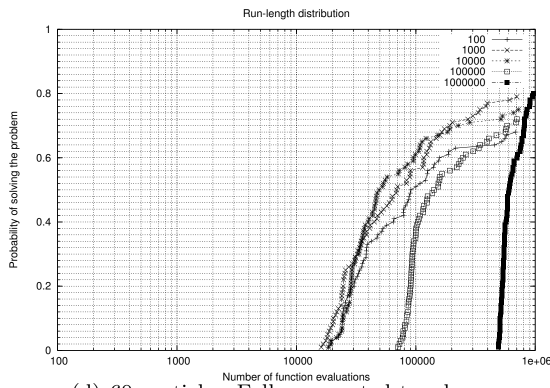
(a) 20 particles, Fully connected topology



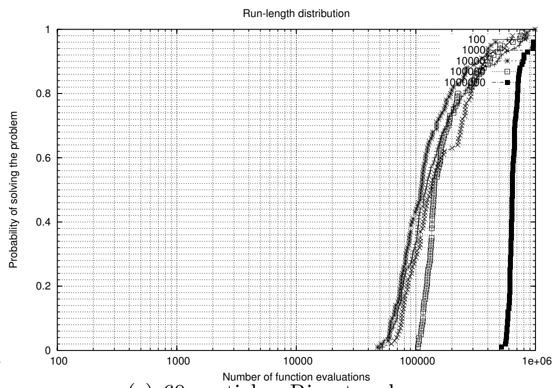
(b) 20 particles, Ring topology



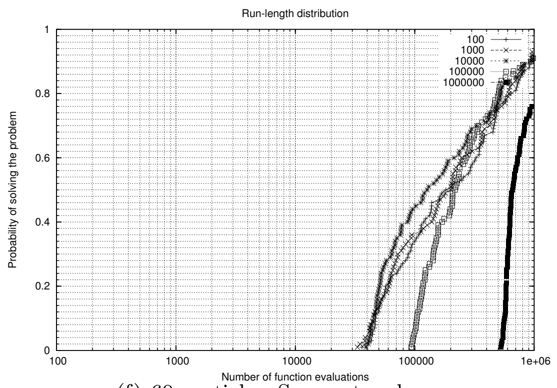
(c) 20 particles, Square topology



(d) 60 particles, Fully connected topology

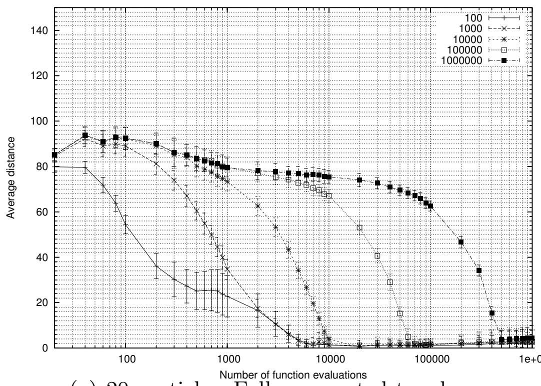


(e) 60 particles, Ring topology

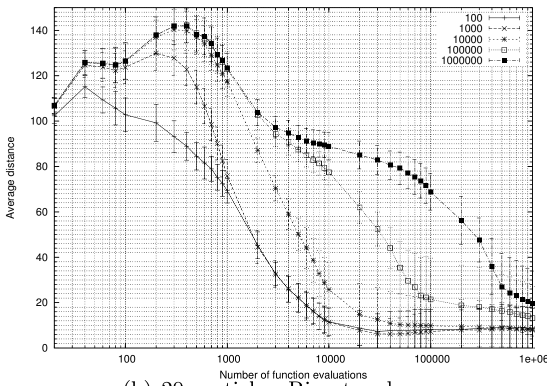


(f) 60 particles, Square topology

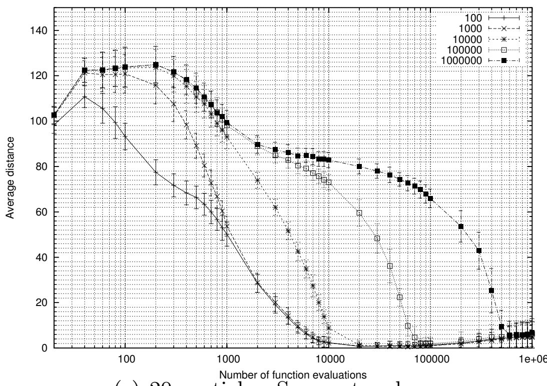
Figure 4.38: Run-length distributions obtained with different inertia weight schedules on the Rosenbrock function. The solution quality demanded is of 10.0% away from the global optimum. Figures (a), (b), and (c) show the run-length distributions obtained with 20 particles. Figures (d), (e), and (f) show the run-length distributions obtained with 60 particles. Columns show the effect of using different population topologies.



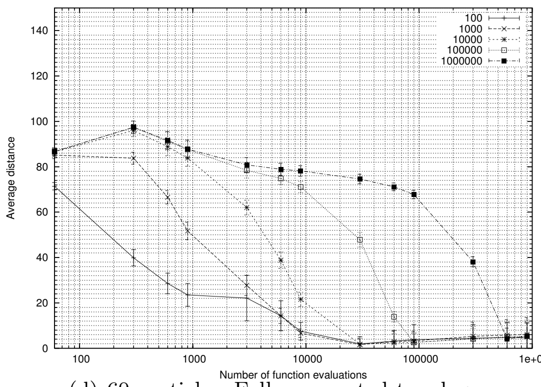
(a) 20 particles, Fully connected topology



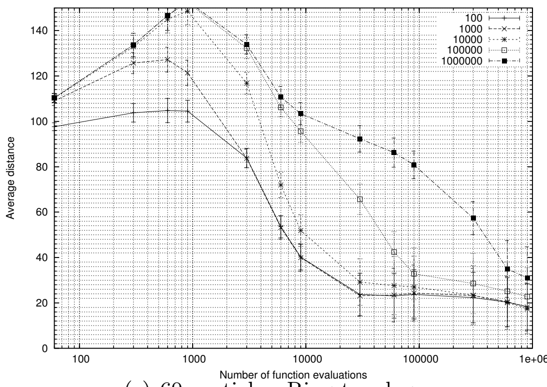
(b) 20 particles, Ring topology



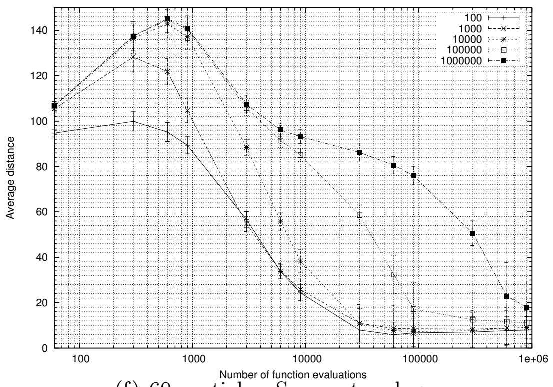
(c) 20 particles, Square topology



(d) 60 particles, Fully connected topology

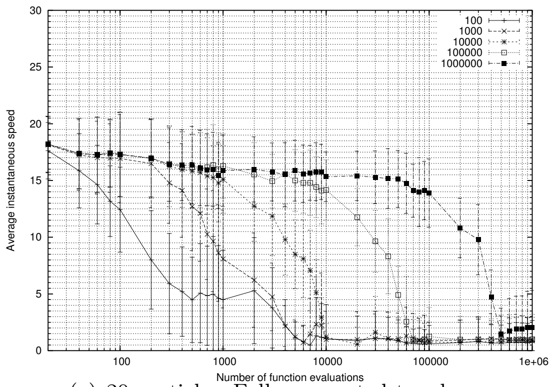


(e) 60 particles, Ring topology

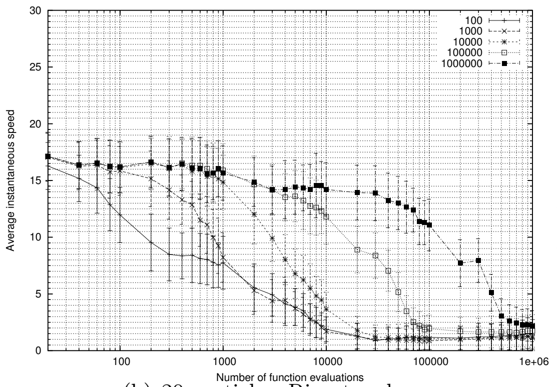


(f) 60 particles, Square topology

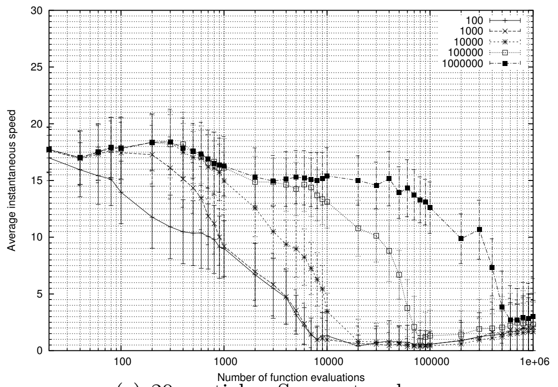
Figure 4.39: Development of the particles' average distance to the swarm's centroid for different inertia weight schedules on the Rosenbrock function. Figures (a), (b), and (c) show the results with 20 particles. Figures (d), (e), and (f) show the results with 60 particles. Columns show the effect of using different population topologies.



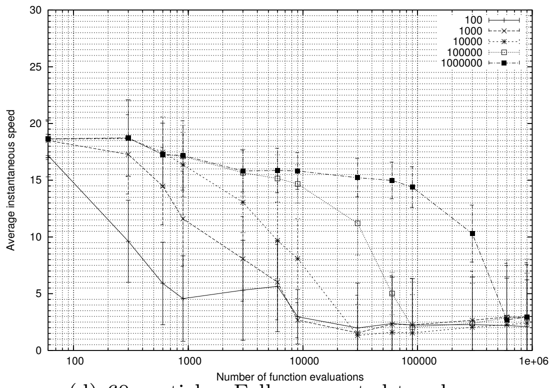
(a) 20 particles, Fully connected topology



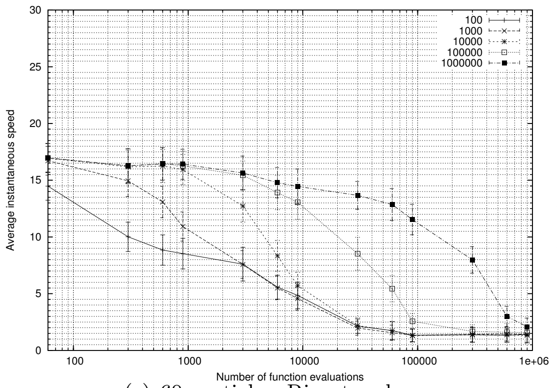
(b) 20 particles, Ring topology



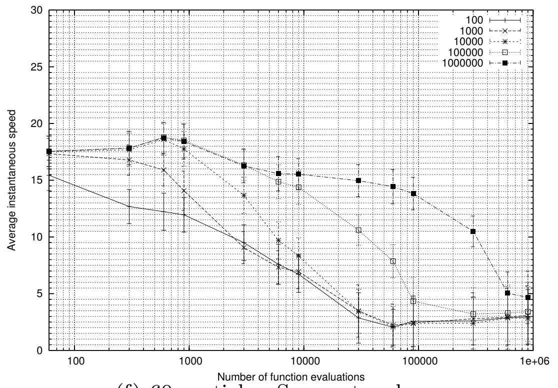
(c) 20 particles, Square topology



(d) 60 particles, Fully connected topology

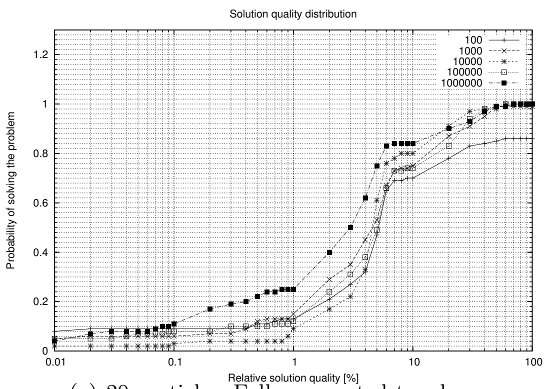


(e) 60 particles, Ring topology

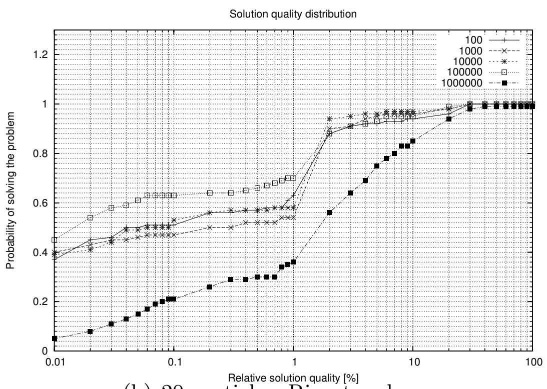


(f) 60 particles, Square topology

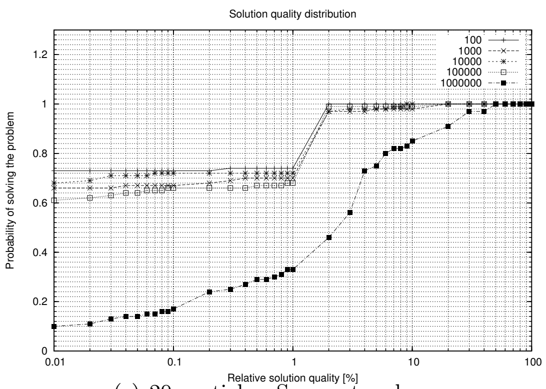
Figure 4.40: Development of the particles' average instantaneous speed for different inertia weight schedules on the Rosenbrock function. Figures (a), (b), and (c) show the results obtained with 20 particles. Figures (d), (e), and (f) show the results obtained with 60 particles. Columns show the effect of using different population topologies.



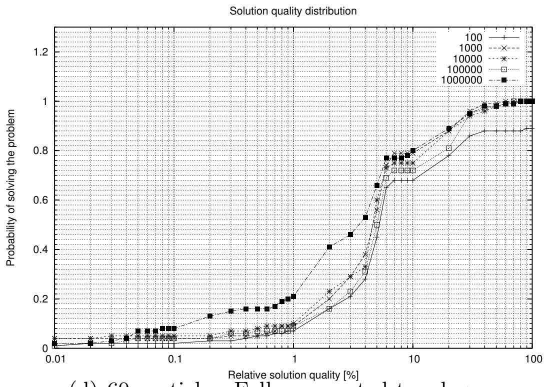
(a) 20 particles, Fully connected topology



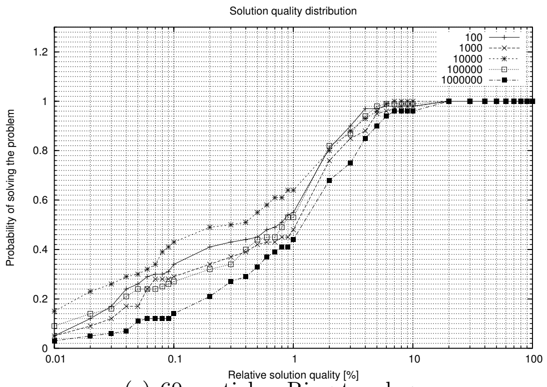
(b) 20 particles, Ring topology



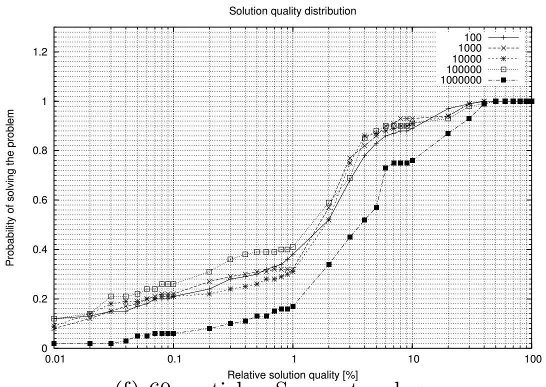
(c) 20 particles, Square topology



(d) 60 particles, Fully connected topology

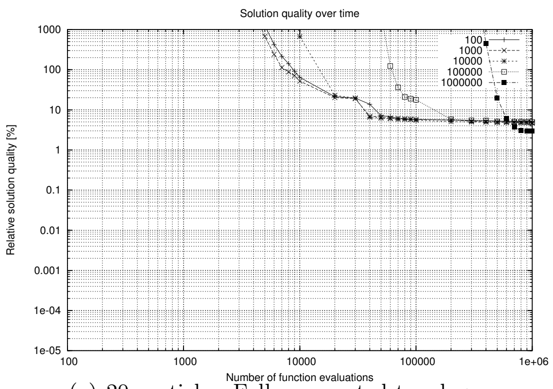


(e) 60 particles, Ring topology

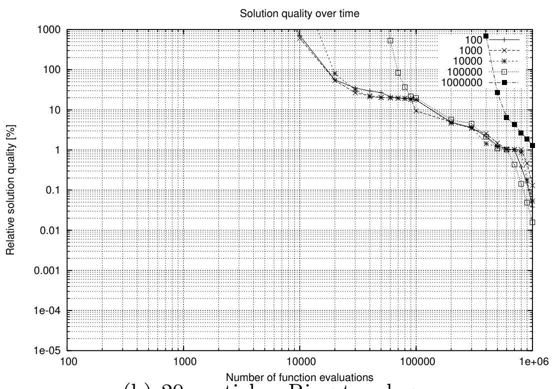


(f) 60 particles, Square topology

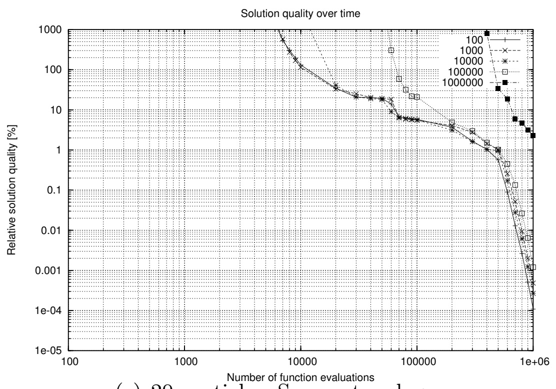
Figure 4.41: Solution-quality distributions for different inertia weight schedules on the Rosenbrock function. These results were obtained after 1 000 000 objective function evaluations. Figures (a), (b), and (c) show the solution-quality distributions obtained with 20 particles. Figures (d), (e), and (f) show the solution-quality distributions obtained with 60 particles. Columns show the effect of using different population topologies.



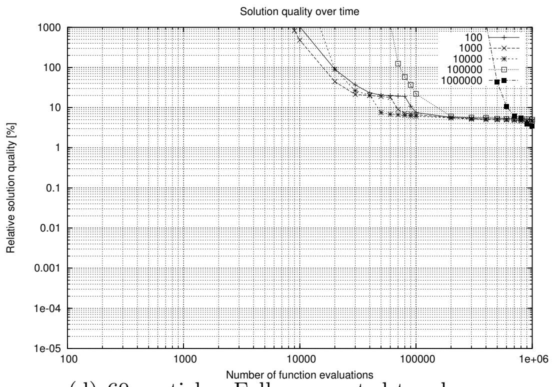
(a) 20 particles, Fully connected topology



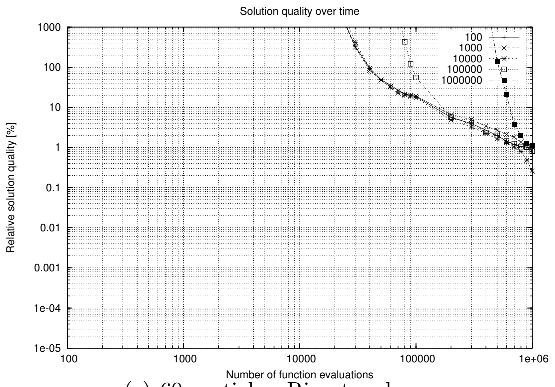
(b) 20 particles, Ring topology



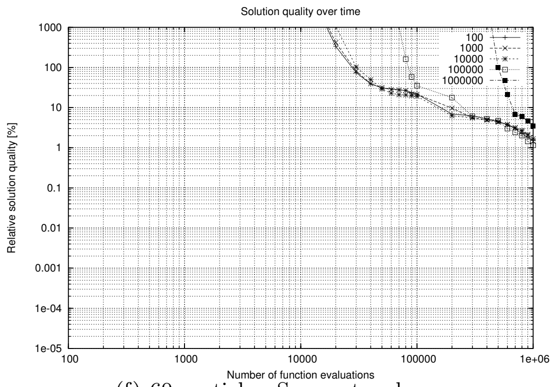
(c) 20 particles, Square topology



(d) 60 particles, Fully connected topology



(e) 60 particles, Ring topology



(f) 60 particles, Square topology

Figure 4.42: Solution quality development over time for different inertia weight schedules on the Rosenbrock function. Figures (a), (b), and (c) show the solution quality development over time that is obtained with 20 particles. Figures (d), (e), and (f) show the solution quality development over time that is obtained obtained with 60 particles. Columns show the effect of using different population topologies. These results are based on medians.

4.4.4 Quantitative Evaluation of the Effects of Different Inertia Weight Schedules in the Time-Decreasing Inertia Weight Particle Swarm Optimizer

We present a comparison of the different possible configurarions of the time-decreasing inertia weight particle swarm optimizer. First, we present an analysis of the sensitivity of the different configurations to a change in the population topology. After that, we present a comparison taking into account all the configurations, to discover which configurations are the best performing and under which conditions. The procedure employed to get the data presented in the following tables and figures was the same we used in Section 4.3. The actual values of the medians are reported in Appendix C on Tables C.1, C.2 and C.3

Table 4.7 shows the relative ranking each configuration gets on each benchmark function. In the rightmost column, the average ranking across all benchmark functions is reported. These data correspond to the case in which the algorithm uses a fully connected topology.

The configuration using the fastest inertia weight schedule is the best ranked one up to 1000 function evaluations. It is ranked as the very best configuration in all benchmark functions except in the case of the Rosenbrock function, in which it is ranked third. For longer runs, it becomes worse and worse ranked, in a nondecreasing way, in almost all functions.

For runs composed of 1 000 000 function evaluations, the best ranked configuration is the one with the slowest inertia weight schedule. It is ranked first (alone or in shared rankings with other configurations) in all benchmark functions except in the Griewank function.

In Figure 4.43(a), in which the average rank obtained by each configuration using the fully connected topology, is plotted against the number of function evaluations, we can see how different schedules are ranked first after different number of function evaluations. For 1000 function evaluations, the best ranked configuration is the one with the schedule in which the inertia weight reaches its minimum value after just 100 function evaluations. For 10 000 function evaluations, the best ranked configuration is the one in which the inertia weight reaches its minimum value after 1000 function evaluations. The pattern continues for the other configurations, but at 1 000 000 function evaluations the best ranked configuration is the one with the slowest schedule. The order in which configurations are ranked at 1000 function evaluations is reversed at 1 000 000 function evaluations. The meaning of this is clear. For short runs, aggressive schedules get the best results, but for long runs, slow schedules get better results. It is a tradeoff between speed and quality.

In Table 4.8 the relative ranking obtained by each configuration, using the square topology, is shown. Similarly to the previous case, the configuration with the fastest schedule is ranked first at 1000 function evaluations in all functions except Rosenbrock. The configuration with the slowest schedule shares the best ranking with other configurations and now, it is relegated to the worst rankings in the Salomon and Rosenbrock functions. In Figure 4.43(b), we see how the configuration with the most aggressive schedule is not inly the best ranked at 1000 function evaluations, but also at 10 000. At 100 000 function evaluations, the best ranked configuration is the one using the next fastest schedule. At 1 000 000 function evaluations, the best ranked configuration is the one with the second slowest schedule.

Table 4.7: Ranking of decreasing inertia weight optimizers with different schedules based on median solution qualities. Fully connected topology case. Each cell contains the rank obtained by the corresponding configuration after 1 000, 10 000, 100 000 and 1 000 000 function evaluations (FES).

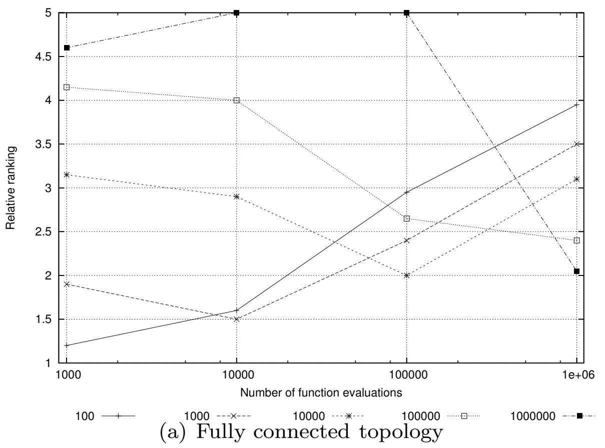
Schedule	Fully connected topology case											
		High dimensional								Low dimensional		
		Multimodal						Unimodal		Multimodal	Unimodal	
	FES	Ackley	Step	Griewank	Rastrigin	Schwefel	Salomon	Sphere	Rosenbrock	Schaffer	Easom	Avg. Rank
100	10^3	1	1	1	1	1	1	1	3	1	1	1.20
	10^4	2	1	1	2	2	1	1	2	2	2	1.60
	10^5	2	2	4	4	4	3	2.5	3	2.5	2.5	2.95
	10^6	4	3	4	5	5	3.5	4	5	3	3	3.95
1000	10^3	2	2	2	2	2	2	2	1	2	2	1.90
	10^4	1	2	2	1	1	2	2	1	1	2	1.50
	10^5	2	2	2.5	3	3	2	2.5	2	2.5	2.5	2.40
	10^6	4	3	2.5	4	4	3.5	4	4	3	3	3.50
10 000	10^3	3	3	3	3	3.5	3	3	2	5	3	3.15
	10^4	3	3	3	3	3	3	3	3	3	2	2.90
	10^5	2	2	2.5	2	2	1	2.5	1	2.5	2.5	2.00
	10^6	4	3	2.5	3	3	3.5	4	2	3	3	3.10
100 000	10^3	4	4.5	4	4	5	4	4	4	3	5	4.15
	10^4	4	4	4	4	4	4	4	4	4	4	4.00
	10^5	4	4	1	1	1	4	2.5	4	2.5	2.5	2.65
	10^6	2	3	1	2	2	3.5	1.5	3	3	3	2.40
1 000 000	10^3	5	4.5	5	5	3.5	5	5	5	4	4	4.60
	10^4	5	5	5	5	5	5	5	5	5	5	5.00
	10^5	5	5	5	5	5	5	5	5	5	5	5.00
	10^6	1	3	5	1	1	1	1.5	1	3	3	2.05

Table 4.8: Ranking of decreasing inertia weight optimizers with different schedules based on median solution qualities. Square topology case. Each cell contains the rank obtained by the corresponding configuration after 1 000, 10 000, 100 000 and 1 000 000 function evaluations (FES).

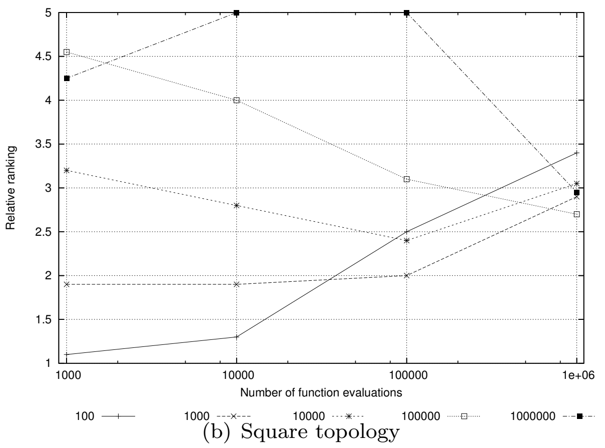
Schedule	Square topology case											
		High dimensional								Low dimensional		
		Multimodal						Unimodal		Multimodal	Unimodal	
	FES	Ackley	Step	Griewank	Rastrigin	Schwefel	Salomon	Sphere	Rosenbrock	Schaffer	Easom	Avg. Rank
100	10^3	1	1	1	1	1	1	1	2	1	1	1.10
	10^4	1	1	1	1	2	1	1	2	1	2	1.30
	10^5	1.5	2	1.5	4	4	2	2	3	2.5	2.5	2.50
	10^6	3	3	2.5	5	5	2.5	3	4	3	3	3.40
1000	10^3	2	2	2	2	2	2	2	1	2	2	1.90
	10^4	2	2	2	2	1	2	2	1	3	2	1.90
	10^5	1.5	2	1.5	3	2	2	2	1	2.5	2.5	2.00
	10^6	3	3	2.5	4	3	2.5	3	2	3	3	2.90
10 000	10^3	3	4	3	3	3	3	3	3	3	4	3.20
	10^4	3	3	3	3	3	3	3	3	2	2	2.80
	10^5	3	2	3	2	3	2	2	2	2.5	2.5	2.40
	10^6	3	3	5	3	4	2.5	3	1	3	3	3.05
100 000	10^3	5	4	4	5	4.5	4	4	5	5	5	4.55
	10^4	4	4	4	4	4	4	4	4	4	4	4.00
	10^5	4	4	4	1	1	4	4	4	2.5	2.5	3.10
	10^6	3	3	2.5	2	2	2.5	3	3	3	3	2.70
1 000 000	10^3	4	4	5	4	4.5	5	5	4	4	3	4.25
	10^4	5	5	5	5	5	5	5	5	5	5	5.00
	10^5	5	5	5	5	5	5	5	5	5	5	5.00
	10^6	3	3	2.5	1	1	5	3	5	3	3	2.95

Table 4.9: Ranking of decreasing inertia weight optimizers with different schedules based on median solution qualities. Ring topology case. Each cell contains the rank obtained by the corresponding configuration after 1 000, 10 000, 100 000 and 1 000 000 function evaluations (FES).

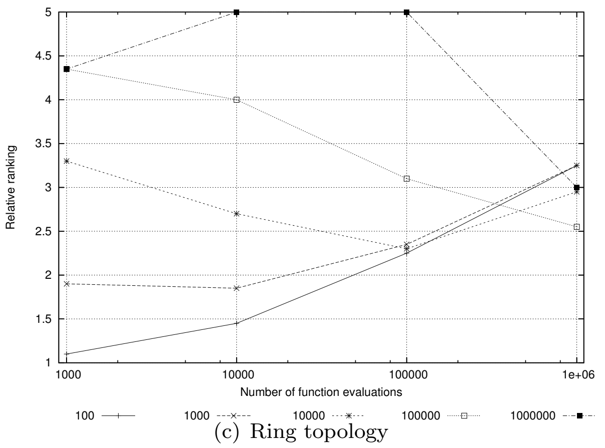
		Ring topology case										
Schedule		High dimensional								Low dimensional		
		Multimodal						Unimodal		Multimodal	Unimodal	
	FES	Ackley	Step	Griewank	Rastrigin	Schwefel	Salomon	Sphere	Rosenbrock	Schaffer	Easom	Avg. Rank
100	10^3	1	1	1	1	1	1	1	2	1	1	1.10
	10^4	1	1	1	2	1	1	1	2	2.5	2	1.45
	10^5	2	1.5	2	4	2	2	2	2	2.5	2.5	2.25
	10^6	3	3	2.5	5	3	3	3	4	3	3	3.25
1000	10^3	2	2	2	2	2	2	2	1	2	2	1.90
	10^4	2	2	2	1	2	2	2	1	2.5	2	1.85
	10^5	1	1.5	2	3	4	2	2	3	2.5	2.5	2.35
	10^6	3	3	2.5	4	5	3	3	3	3	3	3.25
10 000	10^3	3	3	3	3	3	3	3	4	5	3	3.30
	10^4	3	3	3	3	3	3	3	3	1	2	2.70
	10^5	3	3	2	2	3	2	2	1	2.5	2.5	2.30
	10^6	3	3	2.5	3	4	3	3	2	3	3	2.95
100 000	10^3	4	4.5	4	4	5	5	5	3	4	5	4.35
	10^4	4	4	4	4	4	4	4	4	4	4	4.00
	10^5	4	4	4	1	1	4	4	4	2.5	2.5	3.10
	10^6	3	3	2.5	2	2	3	3	1	3	3	2.55
1 000 000	10^3	5	4.5	5	5	4	4	4	5	3	4	4.35
	10^4	5	5	5	5	5	5	5	5	5	5	5.00
	10^5	5	5	5	5	5	5	5	5	5	5	5.00
	10^6	3	3	5	1	1	3	3	5	3	3	3.00



(a) Fully connected topology



(b) Square topology



(c) Ring topology

Figure 4.43: Relative average ranking of decreasing inertia weight optimizers with different schedules. Average ranking over benchmark functions as a function of the number of function evaluations in a run.

In the case of the ring topology, Table 4.9 shows the relative ranking obtained by the different configurations in all benchmark problems. Figure 4.43(c) shows the average ranking obtained by each configuration against the number of function evaluations. Similarly to the case of the square topology, the best ranked configuration at 1000 and 10 000 function evaluations is the one with the fastest schedule. Now, however, it is still the best ranked at 100 000 function evaluations. The best ranked configuration at 1 000 000 function evaluations is, again, the one with the second slowest schedule. The ring topology does not favor convergent behavior and thus, the best ranked configuration is the one with the most aggressive inertia weight schedule.

Table 4.10 shows the average ranking obtained by each configuration at different numbers of function evaluations, and the average ranking across topologies. The standard deviation and, specially, the coefficient of variation indicates the sensitiveness of each configuration to a change in the population topology. The sensitiveness depends on the moment at which it is measured.

There are four cases in which there is no variation in the relative ranking. The configuration with the schedule that reaches the minimum value at 1000 function evaluations, the one whose minimum inertia weight value is reached at 100 000, and the one with the slowest schedule. These configurations are very stable at short, medium, and medium-long runs, respectively.

The configuration with the largest variation is the one with the slowest schedule at 1 000 000 function evaluations. It reflects the fact that with the square and ring topologies, this configuration "has not enough time" to find high quality solutions as a consequence of the boost in the explorative behavior caused by these topologies. In other words, the effect of the convergent behavior induced by the decreasing inertia weight, is exceeded by the explorative behavior induced by a loosely connected topology.

Table 4.11 shows the distribution of appearances of the different configurations in the top 3 group. Three times a configuration appears in the top 3 group in all the functions: twice with the fastest inertia weight schedule, at 1000 and 10 000 function evaluations and once with the second fastest schedule at 10 000 function evaluations. In all cases, except in those using the fastest inertia weight schedule and the second fastest schedule with the fully connected topology, the number of times a configuration appears in the top 3 group increases as the number of function evaluations increases. In the exception cases, the tendency is the inverse, and the frequency of appearance decreases as the number of function evaluations increases.

Figure 4.44 shows the frequency with which each configuration appears in the top 3 group per number of function evaluations. Shade differences represent different population topologies.

On the top-left figure, after 1000 function evaluations, the configurations that dominate are those with the fastest schedules and the fully connected topology.

On the top-right figure, at 10 000 function evaluations, those configurations with the fastest schedules and the fully connected topology continue dominating but, more configurations with square and ring topologies appear in the graph. Important to note is the decreasing pattern (the slower the schedules the lower the frequency of appearance) in configurations using the fully and square topologies.

Table 4.10: Relative average ranking of decreasing inertia weight optimizers with different schedules based on the median solution quality achieved after a certain number of function evaluations. Aggregated results. Each cell contains the rank obtained by the corresponding optimizer after 1 000, 10 000, 100 000 and 1 000 000 function evaluations (FES).

Schedule	FES	Population topology			Average Rank	Standard Deviation	Coefficient of Variation[%]
		Fully Connected	Square	Ring			
100	10^3	1.20	1.10	1.10	1.13	0.06	5.09
	10^4	1.60	1.30	1.45	1.45	0.15	10.34
	10^5	2.95	2.50	2.25	2.57	0.35	13.82
	10^6	3.95	3.40	3.25	3.53	0.37	10.43
1000	10^3	1.90	1.90	1.90	1.90	0.00	0.00
	10^4	1.50	1.90	1.85	1.75	0.22	12.45
	10^5	2.40	2.00	2.35	2.25	0.22	9.69
	10^6	3.50	2.90	3.25	3.22	0.30	9.37
10 000	10^3	3.15	3.20	3.30	3.22	0.08	2.37
	10^4	2.90	2.80	2.70	2.80	0.10	3.57
	10^5	2.00	2.40	2.30	2.23	0.21	9.32
	10^6	3.10	3.05	2.95	3.03	0.08	2.52
100 000	10^3	4.15	4.55	4.35	4.35	0.20	4.60
	10^4	4.00	4.00	4.00	4.00	0.00	0.00
	10^5	2.65	3.10	3.10	2.95	0.26	8.81
	10^6	2.40	2.70	2.55	2.55	0.15	5.88
1 000 000	10^3	4.60	4.25	4.35	4.40	0.18	4.10
	10^4	5.00	5.00	5.00	5.00	0.00	0.00
	10^5	5.00	5.00	5.00	5.00	0.00	0.00
	10^6	2.05	2.95	3.00	2.67	0.53	20.05

Table 4.11: Frequency and distribution of appearances of decreasing inertia weight optimizers with different schedules in the top 3 group.

Schedule	Topology	FES	Ackley	Step	Griewank	Rastrigin	Schwefel	Salomon	Sphere	Rosenbrock	Schaffer	Easom	Count
100	Fully Connected	10^3	*	*	*	*	*	*	*	*	*	*	10
		10^4	*	*	*	*	*	*	*	*	*	*	10
		10^5	*	*	-	-	-	-	*	*	*	*	6
		10^6	-	*	-	-	-	*	*	-	*	*	5
	Square	10^3	*	*	*	*	*	*	*	-	*	*	9
		10^4	*	*	*	*	-	*	*	-	-	*	7
		10^5	*	*	*	-	-	*	*	-	*	*	7
		10^6	*	*	*	-	-	*	*	-	*	*	7
	Ring	10^3	-	-	-	-	*	-	-	-	*	*	3
		10^4	-	-	-	-	-	-	-	-	-	*	1
		10^5	-	-	*	-	-	-	*	-	*	*	4
		10^6	*	*	*	-	-	*	*	-	*	*	7
1000	Fully Connected	10^3	*	*	*	*	-	*	*	*	-	-	7
		10^4	*	*	*	*	*	*	*	*	*	*	10
		10^5	*	*	-	-	-	*	*	*	*	*	7
		10^6	-	*	-	-	-	*	*	-	*	*	5
	Square	10^3	-	-	-	-	-	-	-	-	-	-	0
		10^4	-	-	-	-	*	-	-	*	-	*	3
		10^5	*	*	*	-	-	*	*	-	*	*	7
		10^6	*	*	*	-	-	*	*	-	*	*	7
	Ring	10^3	-	-	-	-	-	-	-	-	-	-	0
		10^4	-	-	-	-	-	-	-	-	-	*	1
		10^5	-	-	*	-	-	-	*	-	*	*	4
		10^6	*	*	*	-	-	*	*	*	*	*	8
10 000	Fully Connected	10^3	-	-	-	-	-	-	-	*	-	-	1
		10^4	-	-	-	-	-	-	-	-	*	*	2
		10^5	*	*	-	-	-	*	*	*	*	*	7
		10^6	-	*	-	-	-	*	*	-	*	*	5
	Square	10^3	-	-	-	-	-	-	-	-	-	-	0
		10^4	-	-	-	-	-	-	-	-	-	*	1
		10^5	*	*	*	*	-	*	*	-	*	*	8
		10^6	*	*	*	-	-	*	*	-	*	*	7

Table 4.11 – continued from previous page

Schedule	Topology	FES	Ackley	Step	Griewank	Rastrigin	Schwefel	Salomon	Sphere	Rosenbrock	Schaffer	Easom	Count
	Ring	10^3	-	-	-	-	-	-	-	-	-	-	0
		10^4	-	-	-	-	-	-	-	-	-	*	1
		10^5	-	-	*	-	-	-	*	-	*	*	4
		10^6	*	*	*	-	-	*	*	*	*	*	8
100 000	Fully Connected	10^3	-	-	-	-	-	-	-	-	-	-	0
		10^4	-	-	-	-	-	-	-	-	-	*	1
		10^5	-	*	-	*	*	-	*	-	*	*	6
		10^6	*	*	-	*	-	*	*	-	*	*	7
	Square	10^3	-	-	-	-	-	-	-	-	-	-	0
		10^4	-	-	-	-	-	-	-	-	-	-	0
		10^5	-	-	-	*	*	-	*	-	*	*	5
		10^6	*	*	*	-	-	*	*	-	*	*	7
	Ring	10^3	-	-	-	-	-	-	-	-	-	-	0
		10^4	-	-	-	-	-	-	-	-	-	*	1
		10^5	-	-	*	-	*	-	-	-	*	*	4
		10^6	*	*	*	-	-	*	*	*	*	*	8
1 000 000	Fully Connected	10^3	-	-	-	-	-	-	-	-	-	-	0
		10^4	-	-	-	-	-	-	-	-	-	-	0
		10^5	-	-	-	-	-	-	-	-	-	-	0
		10^6	*	*	-	*	*	*	*	-	*	*	8
	Square	10^3	-	-	-	-	-	-	-	-	-	-	0
		10^4	-	-	-	-	-	-	-	-	-	-	0
		10^5	-	-	-	-	-	-	-	-	*	*	2
		10^6	*	*	*	*	*	*	*	-	*	*	9
	Ring	10^3	-	-	-	-	-	-	-	-	-	-	0
		10^4	-	-	-	-	-	-	-	-	-	-	0
		10^5	-	-	-	-	-	-	-	-	*	*	2
		10^6	*	*	*	-	*	*	*	-	*	*	8

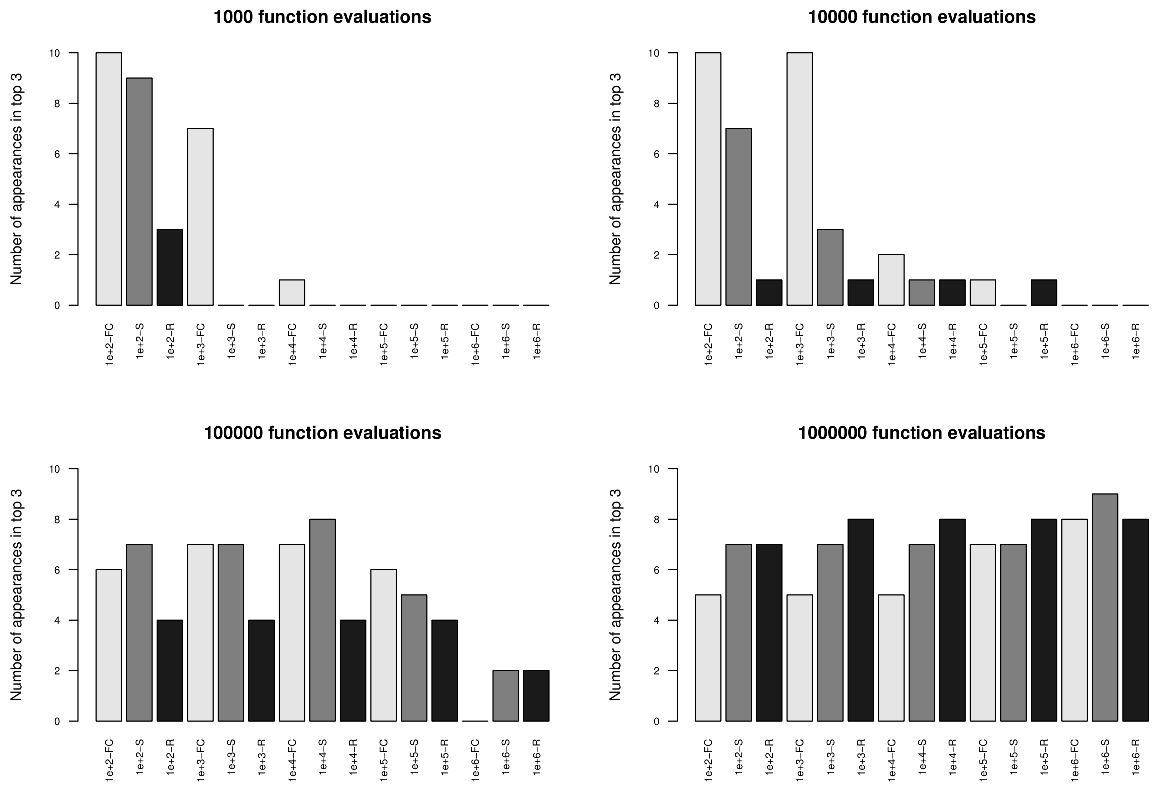


Figure 4.44: Frequency of appearance of decreasing inertia weight optimizers with different schedules in the top 3 group per number of function evaluations. Each figure shows the frequency with which each configuration appears in the top 3 group after some specific number of function evaluations (specified on the top part of each figure). Shade differences represent different population topologies.

On the bottom-left figure, at 100 000 function evaluations, the dominance exhibited by the configurations using a fully connected topology and fast schedules, is lost. There is a significant increase in the frequency of appearance of configurations using the square and ring topologies.

On the bottom-right figure, at 1 000 000 function evaluations, the dominance now is from the configurations with slow schedules and square and ring topologies. Those configurations with fast schedules are among the least frequent in the top 3 group.

To conclude, if it is decided to use the time-decreasing inertia weight particle swarm optimizer and the application requirements impose the need of a limited number of function evaluations, aggressive decreasing inertia weight schedules together with a fully connected topology are the best choice. If the main concern is the solution-quality, moderately slow schedules and moderately connected topologies offer the best results. It depends on the application scenario the compromise to be taken: speed or quality.

4.5 Summary

In Section 4.2, we have presented and discussed the run-length and solution-quality distributions obtained on four problems of our benchmark set. In Section 4.3, we presented a quantitative analysis for discovering the sensitiveness of each particle swarm optimizer to a change in the population topology. We also ranked the 19 variants that result from all the 18 possible combinations between algorithms and topologies (6 algorithms $\times$ 3 topologies), plus AHPSO. This ranking allowed us to identify the best performing particle swarm optimizers considering different function evaluation budgets. We can summarize the main conclusions derived from these two sections as follows:

1. Depending on the problem and the demanded solution quality, different algorithms exhibit a stagnating behavior with different degrees of severity. This stagnating behavior can be alleviated by incrementing the population size or changing the population topology to a loosely connected one. Increasing the population size results in a larger diversity, and thus, it gives the algorithms the opportunity of better exploring the search space. With a loosely connected topology, a particle is not directly and instantaneously influenced by the very best solution found so far and this facilitates their explorative behavior. An increment in the explorative capabilities of an algorithm comes at a price. Even though the probability of solving the problem increases, the speed is slowed down. This is evidenced when, in some problems, the solution quality achieved after a fixed number of function evaluations, drops with larger population sizes or with a loosely connected topology.

   Related points:

   - The number of algorithms using the ring topology in the top 3 group increases as the number of function evaluations increases. This is a sign of the slowness of convergence towards good quality solutions during the first iterations of the algorithms using the ring topology.

   - In many cases, the only particle swarm optimizers that (regardless the number of particles or the population topology) can find, with high probability, the demanded solution qualities are the decreasing inertia weight variant and HPSOTVAC. Of these two variants, HPSOTVAC is faster than the decreasing inertia weight variant (with the inertia weight schedule that reaches its minimum value at 1 000 000 function evaluations). The shapes of the curves of the solution quality development over time for HPSOTVAC and the decreasing inertia weight variant, reveal that they are among the worst particle swarm optimizers during the first 10 000 function evaluations; however, they find some of the best solutions after 1 000 000 function evaluations.

2. Different algorithms are sensitive to a change in the population topology in different degrees. The sensitiveness also depends on the moment in time it is measured.

   Related points:

   - The most sensitive particle swarm optimizer is FIPS. For runs of just a few function evaluations, FIPS's best performance is obtained with the fully connected topology. For longer runs, this algorithm improves its performance as the connectivity of its topology decreases.

3. In situations where the number of function evaluations must be maintained to the minimum, particle swarm optimizers with convergent properties (with highly connected topologies or low inertia weights) get the best results. For medium-long and long runs, algorithms with explorative properties (those with medianly or loosely connected topologies, or large inertia weights) perform best.

   Related points:

   - For medium-long and long function evaluation budgets, all algorithms (except AHPSO, due to its adaptive topology mechanism) benefit from a topology that facilitates an explorative behavior.
   - The overall ranking of the increasing inertia weight variant increases as the connectivity of the topology decreases.
   - The improvement on the ranking obtained by AHPSO is consequence of the exploitation of a convergent behavior caused by a highly connected topology during the first iterations of the algorithm.
   - AHPSO is among the top ranked algorithms in either unimodal or low dimensional problems.

In Section 4.4, we presented the effects of different inertia weight schedules in the decreasing inertia weight particle swarm optimizer. The motivation to carry out these experiments was to see whether we could speed up the convergence of this variant while retaining its high-quality reaching capabilities. We can summarize our conclusions as follows:

1. Faster inertia weight schedules provoke faster decelerations and spatial contractions of the particle swarm.

2. Faster inertia weight schedules provoke the algorithm to exhibit a stronger stagnating behavior. Related points:
    - With slow inertia weight schedules, the algorithm perform poorly during the first function evaluations. However, with them, it is capable to find higher quality solutions.
    - Aggressive inertia weight schedules yield the best results for short runs. For long runs, slow schedules provide the best results. In general, it is a tradeoff between speed and quality.
3. For slow inertia weight schedules, the effect of the convergent behavior induced by the decreasing inertia weight, is exceeded by the explorative induced by a loosely connected topology.

Chapter 5

Improved Particle Swarm Optimizers

In Chapter 4, we evaluated and compared some of the most widely used particle swarm optimization algorithms. The analysis based on run-length and solution-quality distributions provided us with a wealth of information about the behavior of the different algorithms. In this section, we present two improved variants that build on what we learned from that analysis, and that tackle some of the detected weaknesses.

The first variant is one that incorporates an adaptive restart mechanism. It is based on the fact that, in some cases, many short runs are better than a single long run of a stochastic optimization algorithm [17]. The second variant borrows some ideas from recent developments in Ant Colony Optimization [9]. It is a particle swarm optimizer that uses the information gathered throughout the optimization process, to guide the search into the most promising regions of the search space. In both cases, the improvement is evaluated using the same tools we used in the last chapter, that is, using run-length and solution-quality distributions.

5.1 Time-Decreasing Inertia Weight Particle Swarm Optimizer with Adaptive Restarts

In Chapter 3, we described how run-length and solution-quality distributions can reveal the severity of the stagnating behavior exhibited by a stochastic optimization algorithm. In this section, we describe a way to improve the performance of algorithms with strong stagnating behavior without introducing in them any algorithmic modification but controlling the way they are run.

5.2 Effects of Random Restarts on Run-Time Distributions

We will adopt, for a moment, the terminology commonly used in survival analysis and we will refer to the event of finding a solution of the desired quality level as “death” and to the waiting time as “survival” time.

The survival function is then defined as

$$ST_q(t) = 1 - RT_q(t) = P(T_q > t)\,, \tag{5.1}$$

that is, the probability of having to wait more than a time t to find a solution of quality q. Now, imagine an algorithm whose run-time distribution exhibits the following property:

$$P(T_q > t + s | T_q > t) = P(T_q > s)\,, \tag{5.2}$$

where $s > 0$. This property is called *memorylessness*, and it means that, if we have already waited t time without getting the desired solution quality, the probability of having to wait at least s extra time, is the same probability of having to wait an original s time.

If an algorithm is restarted n times, it exhibits this property. To see why this is true, suppose that an algorithm is capable of finding a required solution quality q with probability p at time t. Now, If we restarted the algorithm at this time, a new optimization process would begin from scratch, making the algorithm to indeed “forget” its past.

Let X be a random variable representing the number of restarts needed to find a solution of quality q. If the probability of success of each run is p. Then, the probability that $n = 0, 1, 2, \ldots$ restarts are needed to get one success is

$$P(X = n) = p \cdot (1 - p)^n\,, \tag{5.3}$$

and its cumulative distribution function is defined as

$$P(X \leq n) = 1 - (1 - p)^{n+1}\,. \tag{5.4}$$

To have a clearer picture of the effect of restarts in the performance of a stochastic optimization algorithm, let us consider again the example we used in Chapter 3. Figure 5.1 shows an example of a slowly-increasing run-time distribution. In three of the four desired solution qualities, the algorithm that produced the data for these distributions suffers from a strong stagnating behavior. Note how the probability increases but at an extremely slow rate (recall that the x-axis is in logarithmic scale). The probability of finding a solution of 0.001% away from the optimum is about 0.26 after 10 000 function evaluations, and only 0.37 after 1 000 000 evaluations.

If we restarted the algorithm say, every 10 000 function evaluations, then after just 100 000 function evaluations (i.e., after 10 restarts), we would have a probability of finding the desired solution quality equal to

$$P(X \leq 10) = 1 - (1 - 0.26)^{11} = 0.95\,.$$

Without restarts, we would only achieve a probability of 0.34. It is clear the improvement achieved by just restarting the algorithm.

Only some probability distributions exhibit the memorylessness property: the exponential and geometric distributions. The process described above exhibits a geometric distribution. Indeed, it is an analog of the continuous exponential distribution. Because of this, if we apply restarts to an algorithm with an exponential run-time distribution, no improvement in its performance

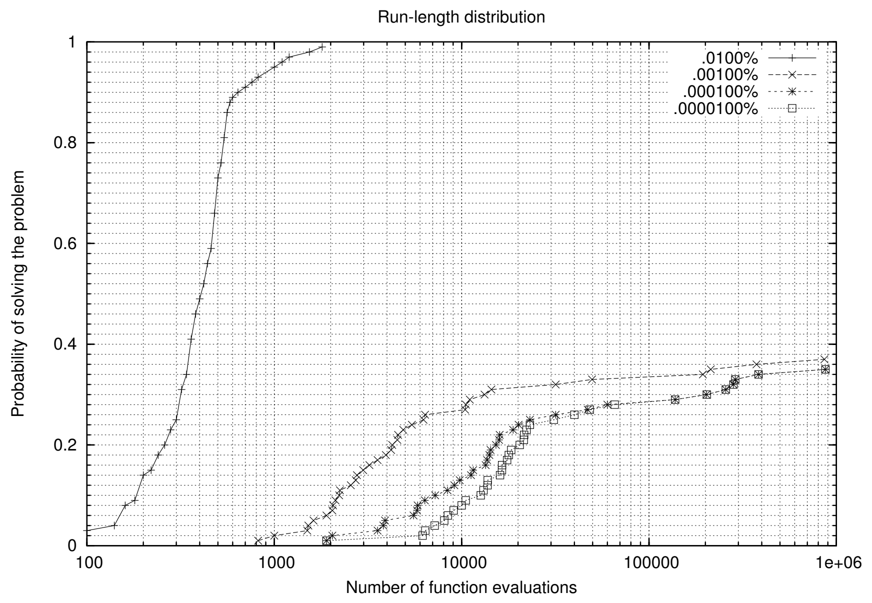


Figure 5.1: Example of a run-length distribution showing an algorithm with strong stagnating behavior. Three out of four solution qualities show how the algorithm that produced these distributions exhibits strong stagnating behavior. Note the logarithmic scale in the x-axis.

is achieved. On the other hand, if an algorithm exhibits a run-time distribution that approaches 1.0 faster than the exponential distribution, using restarts would only deteriorate its performance.

A strategy to avoid the problem of restarting an algorithm with a cumulative distribution function that grows faster than the exponential distribution, is to restart the algorithm only when it starts showing signals of stagnation. In our implementation, we monitor the average swarm's velocity and restart the algorithm when it reaches a level below 1.0% of its maximum allowable speed. We call this strategy *adaptive* to contrast it with a *fixed* restarting strategy that would require us to choose in advance the exact time to restart the algorithm.

5.2.1 Evaluation of Performance

In this section, we present the results we obtained after including an adaptive restart mechanism into the decreasing inertia weight particle swarm optimizer using a greedy configuration (a fully connected topology and a small number of particles). Since the greedy configuration converges faster than any other one, it is with this configuration that the effects of random restarts are better noticed. The inertia weight schedule serves as a way to control the convergence rate and therefore, to see how the adaptive mechanism reacts to different conditions. It is important to stress that the theory behind the effects of restarts on the performance of stochastic optimization algorithms remains valid for other configurations as well.

In the following paragraphs, we present the run-length distributions obtained by the modified algorithm on four of the benchmark problems in our test suite. The results obtained on the rest of the problems and with other configurations are available online at http://iridia.ulb.ac.be/supp/IridiaSupp2006-005/index.html.

Results on the Schaffer Function

Figure 5.2 shows the run-length distributions obtained with different inertia weight schedules and adaptive restarts on the Schaffer function. There are five graphs each corresponding to different inertia weight schedules.

We saw in the last chapter how slow inertia weight schedules do not force particles to reduce their speed in an aggressive way. Due to this fact, the adaptive restart mechanism is never triggered (Figures (d) and (e)). When the algorithm uses faster inertia weight schedules (Figures (a), (b), and (c)), the mechanism is triggered. The result is an increment in the probability of finding the desired solution quality level. The point in time at which the curves start diverging depends on the inertia weight schedule. The faster the schedule, the earlier the deviation of the two curves.

Results on the Rastrigin Function

Figure 5.3 shows the run-length distributions obtained with different inertia weight schedules and adaptive restarts on the Rastrigin function.

In this figure, we can observe the same phenomenon that we described above. The adaptive mechanism seems to properly detect the moment at which the algorithm should be restarted. In the cases of the slowest schedules, the run-length distributions are very steep and the restarting mechanism never gets activated, as expected (Figures (d) and (e)). With the faster schedules (Figures (a), (b), and (c)), the adaptive mechanism "waits" until the algorithm's probability of success starts to decline abruptly.

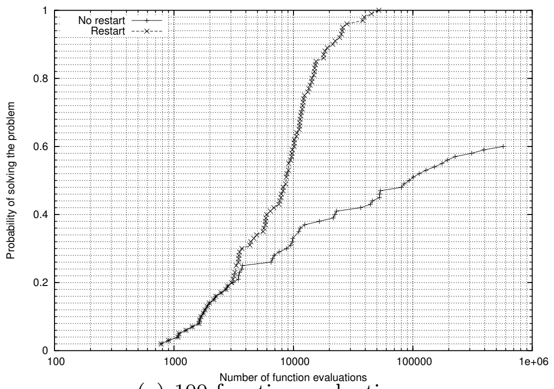
Results on the Sphere Function

In Figure 5.4, where the run-length distributions obtained by the algorithm on the Sphere function are shown, we can see what happens when the adaptive restart mechanism is triggered prematurely. In this case, the algorithm (without restarts) is still capable of finding the required solution quality even after slowing down the particles' speed below the 1% of its maximum allowable value (Figures (a), (b), (c), and (d)). Since this is the criterion for triggering a restart, the result is a deterioration of the algorithm's performance.

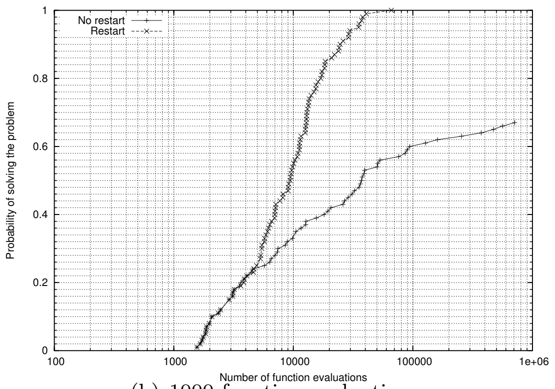
Results on the Rosenbrock Function

Figure 5.5 shows the run-length distributions obtained with different inertia weight schedules and adaptive restarts on the Rosenbrock function.

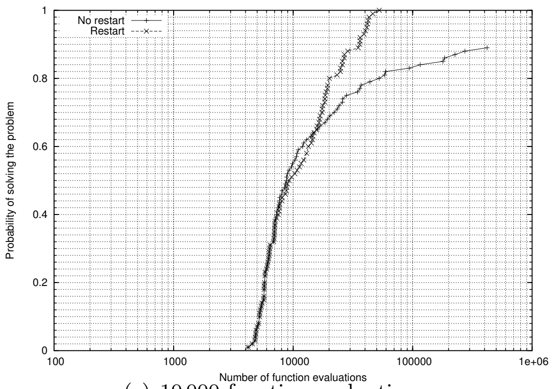
It can be seen that in this case, as was in the case of the Sphere function, restarts are also triggered prematurely. However, in this case, the algorithm (without restarts) stagnates. The version with restarts, although slower, finds a solution with the required solution quality with probability equal to 1.0. It is, therefore, evident that even if the restart mechanism is triggered prematurely, the resultant algorithm is capable, may be in the long run, of finding a solution of the required quality.



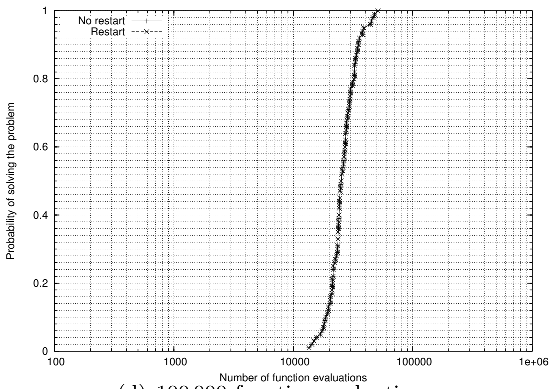
(a) 100 function evaluations



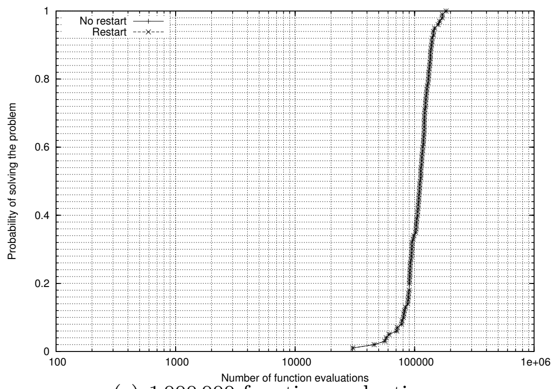
(b) 1000 function evaluations



(c) 10 000 function evaluations

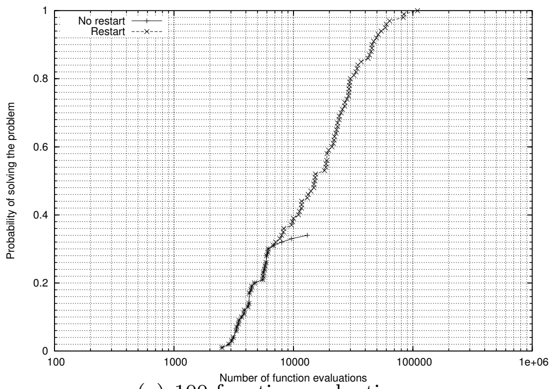


(d) 100 000 function evaluations

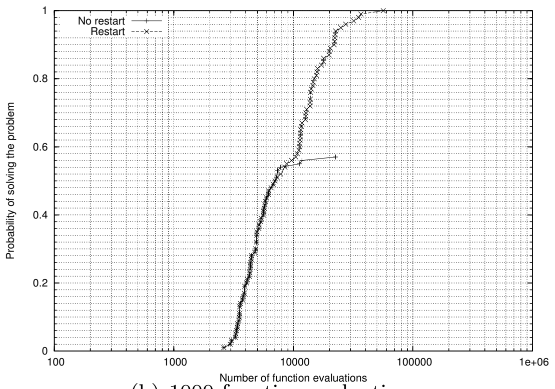


(e) 1 000 000 function evaluations

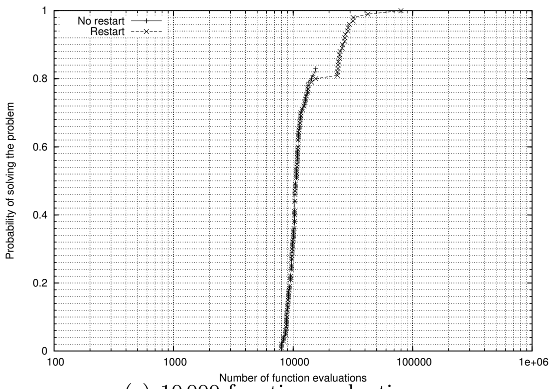
Figure 5.2: Run-length distributions obtained with different inertia weight schedules and adaptive restarts on the Schaffer function. The solution quality demanded is of 0.0001% away from the global optimum. The succession of graphs shows the effects of the adaptive restart mechanism in the performance of the decreasing inertia weight optimizer with different inertia weight schedules.



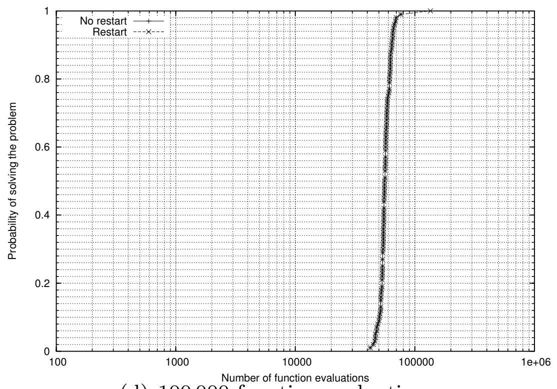
(a) 100 function evaluations



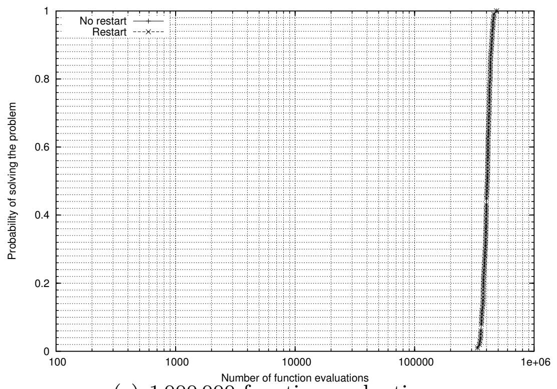
(b) 1000 function evaluations



(c) 10 000 function evaluations

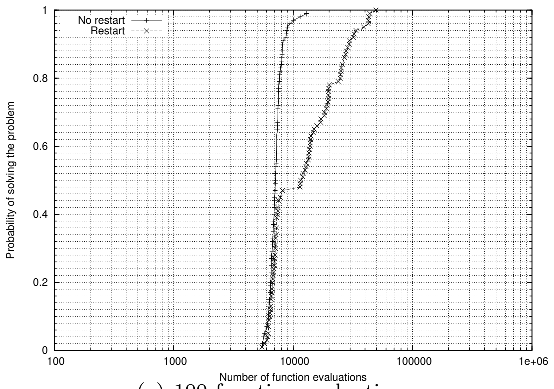


(d) 100 000 function evaluations

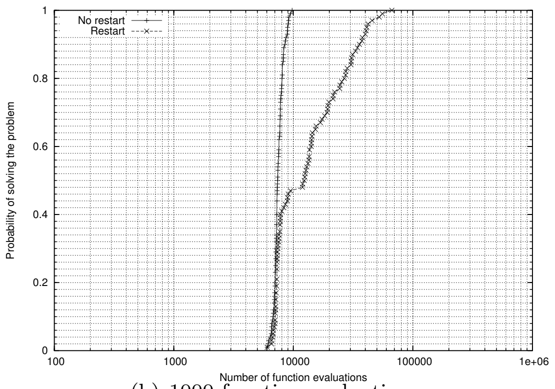


(e) 1 000 000 function evaluations

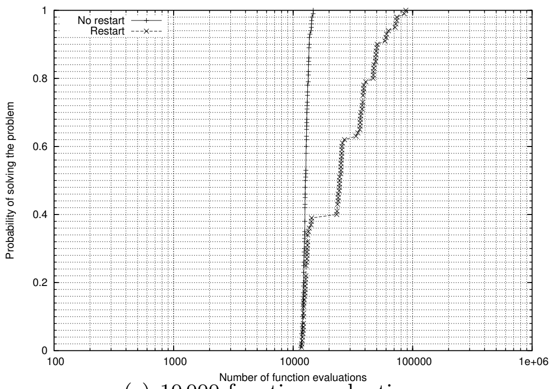
Figure 5.3: Run-length distributions obtained with different inertia weight schedules and adaptive restarts on the Rastrigin function. The solution quality demanded is of 20.0% away from the global optimum. The succession of graphs shows the effects of the adaptive restart mechanism in the performance of the decreasing inertia weight optimizer with different inertia weight schedules.



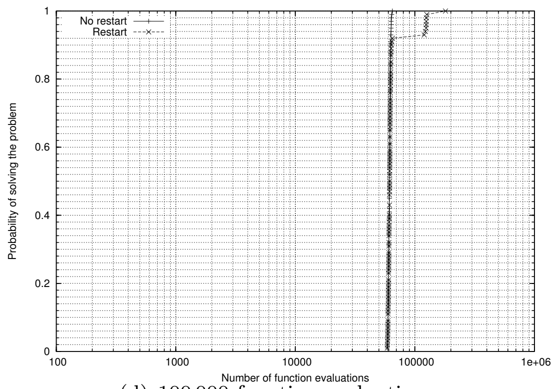
(a) 100 function evaluations



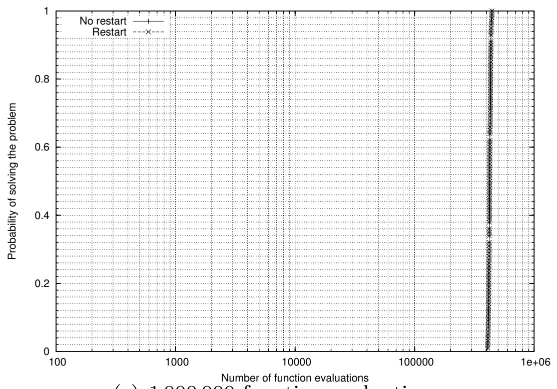
(b) 1000 function evaluations



(c) 10 000 function evaluations

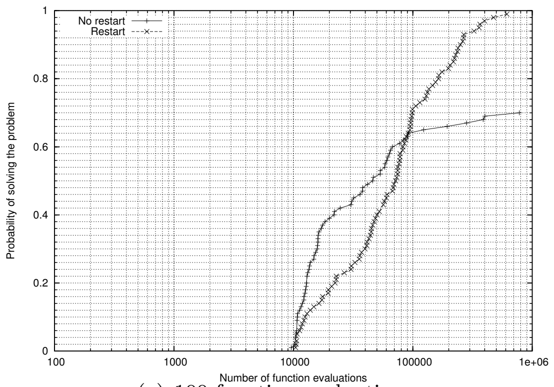


(d) 100 000 function evaluations

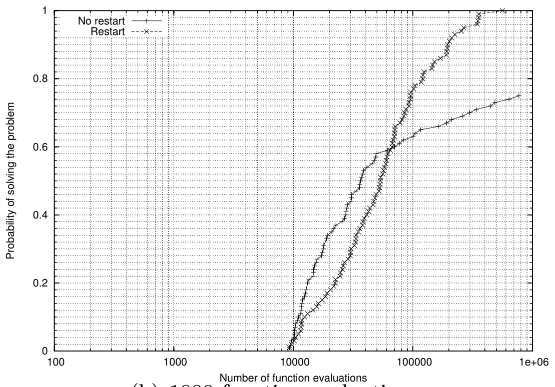


(e) 1 000 000 function evaluations

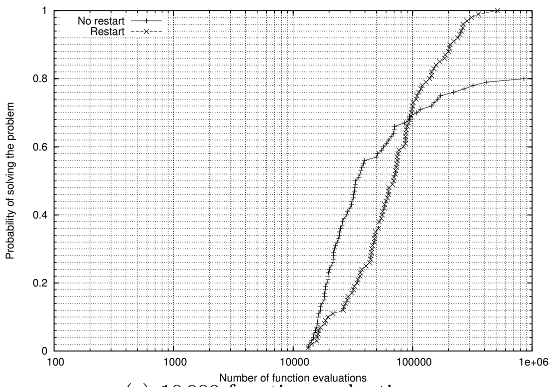
Figure 5.4: Run-length distributions obtained with different inertia weight schedules and adaptive restarts on the Sphere function. The solution quality demanded is of 0.01% away from the global optimum. The succession of graphs shows the effects of the adaptive restart mechanism in the performance of the decreasing inertia weight optimizer with different inertia weight schedules.



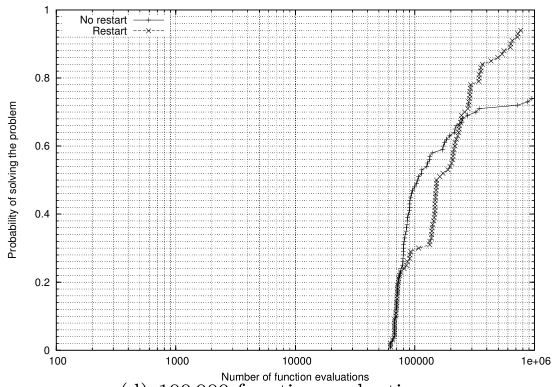
(a) 100 function evaluations



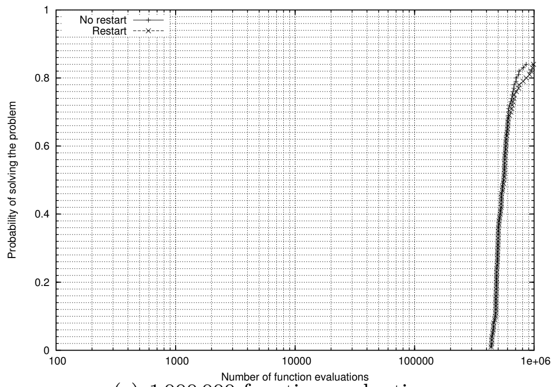
(b) 1000 function evaluations



(c) 10 000 function evaluations



(d) 100 000 function evaluations



(e) 1 000 000 function evaluations

Figure 5.5: Run-length distributions obtained with different inertia weight schedules and adaptive restarts on the Rosenbrock function. The solution quality demanded is of 10.0% away from the global optimum. The succession of graphs shows the effects of the adaptive restart mechanism in the performance of the decreasing inertia weight optimizer with different inertia weight schedules.

The idea of applying restarts in stochastic optimization algorithms is not new. It has been common practice in the mainstream Evolutionary Algorithms field (see, e.g. [14, 18, 28, 29]). However, to the best of our knowledge, it has not been applied in Particle Swarm Optimization before. More importantly, however, is that we showed how run-length and solution quality distributions, serve as analytic tools that help us choose or devise improvement strategies.

Choosing the optimal restart schedule is not trivial. One needs to know the run-length distribution of the algorithm one wants to use on the problem of interest. However, in practical applications, estimating the run-length distribution makes no sense at all, since it would imply to solve the problem repeatedly and our goal is normally to solve it just once. An adaptive restart strategy monitors performance of the algorithm and restarts it if some criteria are met. This is a much more flexible approach that can be easily used in practice.

The major problem of adaptive restart strategies is that, as we saw in our results, it depends on the problem being solved. In other words, what works fine for solving some problems might not work at all for some others.

One of the major implications of our results is that by applying restarts to an algorithm, its performance can be improved significantly. This fact should be taken into account when comparing algorithms and when choosing one for solving a particular problem.

5.3 Estimation of Distribution Particle Swarm Optimizer

In the classical particle swarm optimization algorithm (see Section 2.2), particles exploit their individual memory and that of their neighbors, to explore the search space. However, the swarm as a whole has no means to exploit its collective memory (represented by the set of vectors $\vec{pbest}_i$, that represent the particles' previous best positions) to guide its search. We believe this causes a re-exploration of already known bad regions of the search space, wasting costly function evaluations. In our approach, we use the swarm's collective memory to probabilistically guide the particles' movement towards the estimated promising regions in the search space. In this way, many function evaluations could be saved from being wasted.

In Chapter 2, where we presented the particle swarm optimization approach and some algorithmic variants, we saw how the behavior of every particle is partially determined by its previous experience (through vector $\vec{pbest}_i$). This memory allows a particle to search somewhere around its own previous best position and the best position ever found by a particle in its neighborhood. However, during a search different particles move and test (i.e., evaluate the objective function) over and over again the same, or approximately the same, region in the search space without producing any individual improvement. While this is part of the search process and allows the swarm to explore the search space, it is also a waste of computing power when the explored regions have been visited before by the swarm without success. This happens because the swarm as a single entity is memoryless.

In this section, we present a generic extension to the Particle Swarm Optimization paradigm that allows a particle swarm to estimate the distribution of promising regions (and thus, “learn” from previous experience) of the fitness landscape by exploiting the information it gains during the optimization process. This distribution is in turn used to try to keep the particles within the promising regions. It is a modular extension that can be used in any particle swarm variant that uses a position update rule based on previously found solutions. The estimation of the distribution is done by means of a mixture of normal distributions taking into account the set of *pbest* vectors. It borrows some ideas from recent developments in Ant Colony Optimization [9], in which an archive of solutions is used to select the next point to explore in the search space. The underlying assumption of independence between variables common to many Estimation of Distribution Algorithms (EDAs) for continuous optimization problems [38] is also present in our approach.

The following section briefly presents some background information about EDAs and establishes the relationship between our approach and pure EDAs.

5.3.1 Estimation of Distribution Optimization Algorithms

Evolutionary Algorithms that use information obtained during the optimization process to build probabilistic models of the distribution of good regions in the search space and that use these models to generate new solutions are called Estimation of Distribution Algorithms (EDAs) [38]. The fully joint probability distribution characterizes the problem being solved. Depending on whether there is *a priori* knowledge about the underlying distribution or not, one can either use a suitable parameterization to get fast convergence rates or use machine learning methods to approximate this unknown distribution, respectively. The latter case is the most commonly found in practice.

EDAs differ in three fundamental aspects: (i) in the way they gather information during the optimization process, (ii) in the way they use the gathered information to build probabilistic models, and (iii) in the way they use these models to generate new solutions. An experimental comparison of some of the best known EDAs has been done by Kern et al. [26].

A pseudo-code view of the algorithmic structure behind most EDAs can be seen in Algorithm 2. An EDA starts with a solution population $\mathbf{X}^0$ and a solution distribution model $\mathcal{P}^0$. The main loop consists of four principal stages. The first stage is to select the best individuals (according to some fitness criteria f) from the population. These individuals are used in a second stage in which the solution distribution model $\mathcal{P}^t$ is updated or recreated. The third stage consists of sampling the updated solution distribution model $\mathcal{P}^{t+1}$ to generate new solutions $\mathbf{X}^{t+1}_{offspring}$. The last stage involves the base population $\mathbf{X}^t_{base}$, the set of new solutions $\mathbf{X}^{t+1}_{offspring}$ and the fitness criteria f. The end result of this final stage is a new base population. The process starts over again until the stopping criteria are satisfied.

There has been a growing interest for EDAs in the last years and there are now some hybrid approaches. One of them is our proposed algorithm. It is a particle swarm optimizer that uses the solution distribution model $\mathcal{P}^t$, to keep the particles within the promising regions so that it does not waste function evaluations. It is not a pure EDA because it does not generate new solutions from the solution distribution model unless it is needed. We detail our algorithm in the next section.

Algorithm 2 Algorithmic structure of Estimation of Distribution Algorithms.

```
{Initialization}
Initialize population of solutions $\mathbf{X}_{base}^{0}$ and solution distribution model $\mathcal{P}^0$

{Main Loop}
while Stopping criteria are not satisfied do
    $\mathbf{X}_{parent}^{t} = select(\mathbf{X}_{base}^{t}, f)$ {Selection}

    $\mathcal{P}^{t+1} = estimate(\mathbf{X}_{parent}^{t}, \mathcal{P}^t)$ {Estimation}

    $\mathbf{X}_{offspring}^{t+1} = sample(\mathcal{P}^{t+1})$ {Sampling}

    $\mathbf{X}_{base}^{t+1} = replacement(\mathbf{X}_{offspring}^{t+1}, \mathbf{X}_{base}^{t}, f)$ {Replacement}

    $t = t + 1$
end while
```

5.3.2 Estimation of Distribution Particle Swarm Optimization Algorithm

Particle swarm optimization algorithms are considered to be part of the emerging field of *Swarm Intelligence* [24] [5]. Swarm Intelligence is the discipline that studies natural and artificial systems composed of multiple simple entities that collectively exhibit adaptive behaviors. Some examples of natural swarm intelligent systems are ant colonies, slime molds, bee and wasp swarms.

Besides Particle Swarm Optimization, the other prominent representative of artificial swarm intelligent systems is Ant Colony Optimization [9]. It is usually used for solving combinatorial optimization problems. In ant colony optimization algorithms, artificial ants build solutions incrementally selecting one solution component at a time. The probabilistic selection is biased by a trail of "pheromone" deposited by other ants in previous iterations of the algorithm. The amount of pheromone is proportional to the quality of the complete solutions, so that ants will prefer to choose solution components that are known to yield good solutions. In fact, the role of the so-called pheromone matrix is to approximate the distribution of good solutions in the search space. Seen from this point of view, ant colony optimization algorithms belong to the family of estimation of distribution algorithms.

A recent development of Ant Colony Optimization that extends it to continuous optimization is called ACO$_R$ [51] [52]. ACO$_R$ approximates the joint probability distribution, one dimension at a time, by using mixtures of weighted Gaussian functions. The weights represent the quality of different search regions. This allows the algorithm to deal with multimodal functions. Figure 5.6 illustrates the idea of approximating the distribution of good regions in a single dimension using a mixture of weighted Gaussian functions.

In ACO$_R$, the source of information to parameterize these univariate distributions is an archive of solutions of size k. The i-th component of the l-th solution is denoted by s_l^i. For an n-dimensional problem, $1 \leq i \leq n$ and $1 \leq l \leq k$. For each dimension i, the vector $\vec{\mu}_i =< s_1^i, \dots, s_k^i >$ is the vector of means that is used to model the univariate probability distribution of the i-th

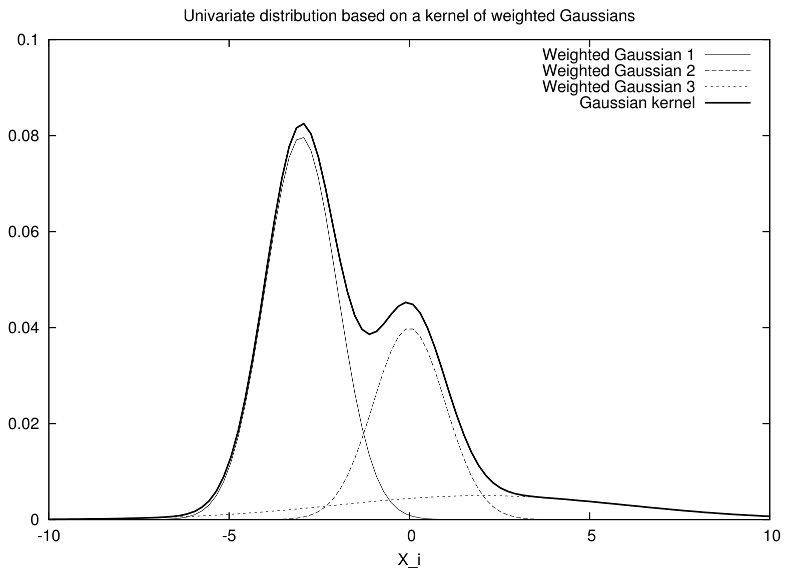


Figure 5.6: Mixture of weighted one-dimensional Gaussian functions used to approximate two promising (but in a different degree) search regions.

dimension. The vector of weights $\vec{w} =< w_1, \ldots, w_k >$ is the same across all dimensions because it is based on the relative quality of the complete solutions. Every iteration, after the solutions are ranked, the weights are determined by

$$w_l = \frac{1}{qk\sqrt{2\pi}} e^{-\frac{(l-1)^2}{2(qk)^2}}\,, \tag{5.5}$$

where $q > 0$ is a parameter that determines the degree of preferability of good solutions. The smaller q, the stronger the preference of the best solutions to guide the search.

Since ACO$_R$ samples the mixture of Gaussians, it has to first select one of the Gaussian functions from the kernel. The selection is done probabilistically. The probability of choosing the l-th Gaussian function is proportional to its weight and it is computed using

$$p_l = \frac{w_l}{\sum_{j=1}^{k} w_j}\,. \tag{5.6}$$

Then, ACO$_R$ computes the standard deviation of the chosen Gaussian function as

$$\sigma_l^i = \xi \sum_{j=1}^{k} \frac{|s_j^i - s_l^i|}{k-1}\,, \tag{5.7}$$

where $\xi > 0$ is a parameter that controls the wideness of the chosen Gaussian function. The smaller ξ, the narrower the range over which ACO$_R$ searches. ξ has the same value for all the dimensions.

Having computed all the needed parameters, ACO$_R$ samples the Gaussian function to generate a new solution component. The process is repeated for every dimension, for every ant until a stopping criterion is met.

This fast presentation of ACO$_R$ was needed to introduce our Estimation of Distribution Particle Swarm Optimization (EDPSO) algorithm. The reason is that EDPSO borrows some ideas from ACO$_R$.

The fundamental data structure of EDPSO is the set of vectors $\vec{pbest}_i$, that represent the particles' previous best positions. This set plays the role of the solution archive in ACO$_R$. It is though this set that the swarm accumulates its experience in the search process, so we use this experience to guide the future of the search. In EDPSO, the parameter k (i.e., the size of the solution archive) is equal to the number of particles.

EDPSO works as a canonical PSO as described in Section 2.3.1 but with some modifications: after the execution of the velocity update rule shown in Equation 2.4, EDPSO selects one Gaussian function just as ACO$_R$ does. Then, the selected Gaussian function is evaluated (not sampled) to probabilistically move the particle to its new position. The value returned by the evaluation of the Gaussian function is interpreted as a probability of accepting a particles's new position. If the movement is accepted, the algorithm continues as usual, but if the movement is not accepted, then the selected Gaussian function is sampled in the same way as in ACO$_R$.

The result is a hybrid algorithm that explores the search space using the usual Particle Swarm Optimization dynamics but when this approach fails (i.e., when a particle's tendency is to move far away from good solutions) a direct sampling of the probability distribution is used instead. It is important to mention that when the selected Gaussian function is evaluated, we use an *unscaled* version of it, so that its range is [0,1] (i.e., a true probability). A pseudo-code version of EDPSO is presented in Algorithm 3.

5.3.3 Evaluation of Performance

The evaluation of the performance of EDPSO was carried out using the same 10 benchmark functions presented in Section 4.1.1. Since EDPSO is based on the canonical particle swarm optimizer, its non-free parameters were set as described in Table 4.1. The algorithm was run 100 times on each problem for a maximum of 1 000 000 function evaluations.

The free parameters were the number of particles, the swarm's population topology, and parameters q and ξ. For the first two parameters, we tried three different population sizes: 20, 40 and 60 particles, and three topologies: fully connected topology, ring topology and a square topology. The sets of values we tried for q and ξ were $\{0.1, 0.2\}$ and $\{0.85, 1.0\}$, respectively.

The run-length and solution-quality distributions we present in the following paragraphs were obtained using swarms of 20 and 60 particles. We focus our attention to the distributions obtained on three benchmark functions only. The run-length and solution-quality distributions presented in this section correspond to a specific solution quality and a certain maximum number of function evaluations. The results obtained in the rest of the benchmark functions are available online at http://iridia.ulb.ac.be/supp/IridiaSupp2006-005/index.html.

Results on the Griewank Function

Figure 5.7 shows the run-length distributions obtained by EDPSO for a solution quality of 0.001% away from the optimum. Recall that the Griewank function is

Algorithm 3 Pseudocode version of the EDPSO algorithm

```
{Initialization. $k$ is the number of particles, and $n$ is the dimension of the problem}
for $i = 1$ to $k$ do
  Create particle $i$ with random position and velocity
end for
Initialize $gbest$ and all $pbest_i$ to some sensible values {To a sufficiently large number, for example, if we want to minimize a function}

{Main Loop}
$t = 0$
while $gbest$ is not good enough or $t < t_{max}$ do
  {Evaluation Loop}
  for $i = 1$ to $k$ do
    if $f(\vec{x}_i)$ is better than $pbest_i$ then
      $\vec{p}_i = \vec{x}_i$
      $pbest_i = f(\vec{x}_i)$
    end if
    if $pbest_i$ is better than $gbest$ then
      $gbest = pbest_i$
      $\vec{s} = \vec{p}_i$
    end if
  end for
  {Update Loop}
  Rank all $pbest_i$ according to their quality
  Compute $\vec{w} =< w_1, \ldots, w_k >$ using Equation 5.5
  Compute all $p_l$ using Equation 5.6
  for $i = 1$ to $k$ do
    for $j = 1$ to $n$ do
      $v_{ij} = \chi(v_{ij} + \varphi_1 U_1(0,1)(p_{ij} - x_{ij}) + \varphi_2 U_2(0,1)(s_{ij} - x_{ij}))$
      $x_{ij}^{candidate} = x_{ij} + v_{ij}$
      Select a Gaussian function from the kernel according to $p_l$, name it $g_l^i$.
      Compute $\sigma_l^i$ using Equation 5.7
      $prob_{move} = \sigma_l^i \sqrt{2\pi} g_l^i(x_{ij}^{candidate})$ {$\sigma_l^i \sqrt{2\pi}$ unscales the function}
      if $U_3(0,1) < prob_{move}$ then
        $x_{ij} = x_{ij}^{candidate}$ {The particle moves normally}
      else
        $x_{ij} = sample(g_l^i)$ {New position is a sample from the chosen function}
      end if
    end for
  end for
  $t = t + 1$
end while
```

the only one in which all the particle swarm optimizers compared in Chapter 4 (with 20 particles) could not find the required solution quality with probability equal to one. EDPSO behaves in the same way.

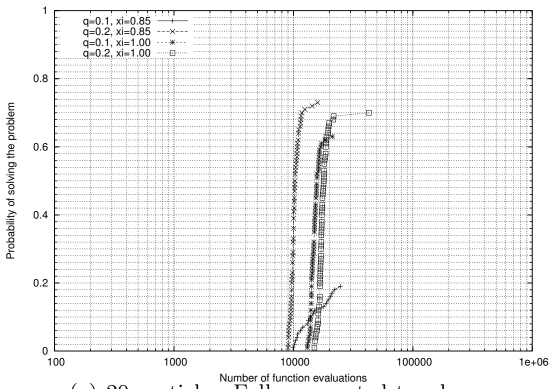
It is easier to analyze the behavior of EDPSO is we first focus on the cases in which parameter $\xi = 0.85$. In this case, we see how with 20 particles and $q = 0.1$, the algorithm suffers from severe stagnating behavior. With $q = 0.2$, however, the performance improves considerably. No qualitative change in the behavior of these two configurations can be observed after a change of the population topology. If we increase the number of particles, all configurations improve their performance. The configurations with $\xi = 1.0$ are, in general, slower than the ones using $\xi = 0.85$. If we compare Figures 4.18 and 5.7 we see how, in spite of the stagnating behavior exhibited by EDPSO, at least one of its configurations finds the required solution quality with greater probability than any of the optimizers compared in Chapter 4. The only exception is FIPS with 60 particles and the ring topology, which finds a solution of the required quality faster than any EDPSO configuration.

Figure 5.8 shows the solution-quality distributions obtained by the different EDPSO configurations after 1 000 000 function evaluations. The greediest EDPSO configuration ($\xi = 0.85$, $q = 0.1$) is the worst performing in the 20 particles case. With 60 particles, all configuration reach solution qualities of 0.01% away from the optimum with probability equal to one, irrespective to the population topology used. With 20 particles (excluding the results of the greediest configuration), the lowest probability of finding a solution of quality 0.01% away from the optimum is approximately equal to 0.9. In Figure 4.19 we can see how no algorithm, using 20 particles, was capable of reaching this solution quality probability with a probability greater than 0.84, irrespective of the population topology.

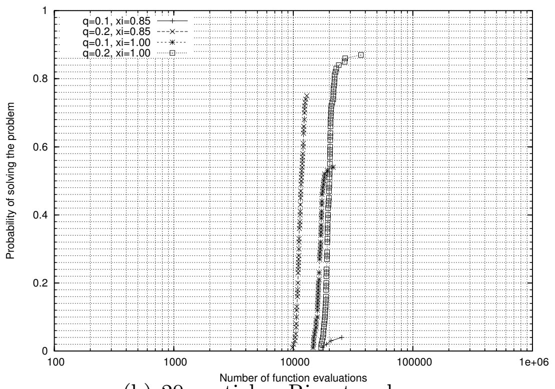
Figure 5.9 shows the median solution quality development over time achieved by the different EDPSO configurations. The fastest configuration of all is the one with 20 particles, the fully connected topology, $\xi = 0.85$ and $q = 0.2$. With 60 particles, there's a clear separation between those configurations with $\xi = 0.85$ and those with $\xi = 1.0$. In this case, there is no significant difference between the configurations using $q = 0.1$ and $q = 0.2$.

Results on the Rastrigin Function

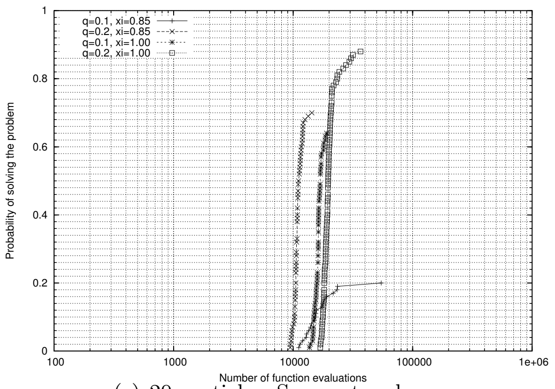
Figure 5.10 shows the run-length distributions obtained by EDPSO for a solution quality of 20.0% away from the optimum. With 20 particles, the only configuration that does not find a solution of this quality with a high probability is the greediest one. From the other three configurations, the fastest is the one with $\xi = 0.85$ and $q = 0.2$. However, with the square and ring topologies, it is not distinguishable from the one with $\xi = 1.0$ and $q = 0.1$. With 60 particles, there is a significant slow down of the algorithm's performance. In comparison with other particle swarm optimizers, EDPSO's performance can be compared with that of HPSOTVAC which was the best optimizer for solving the Rastrigin function.



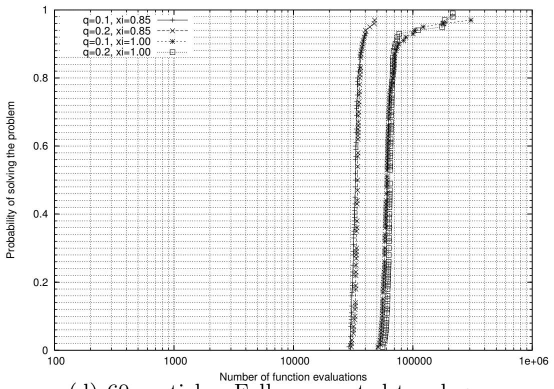
(a) 20 particles, Fully connected topology



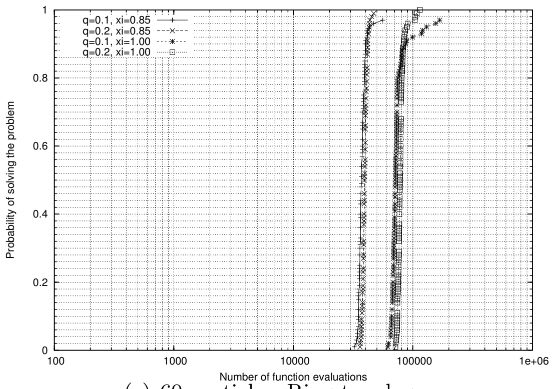
(b) 20 particles, Ring topology



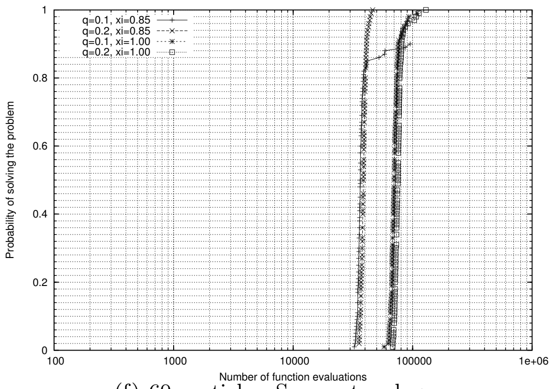
(c) 20 particles, Square topology



(d) 60 particles, Fully connected topology

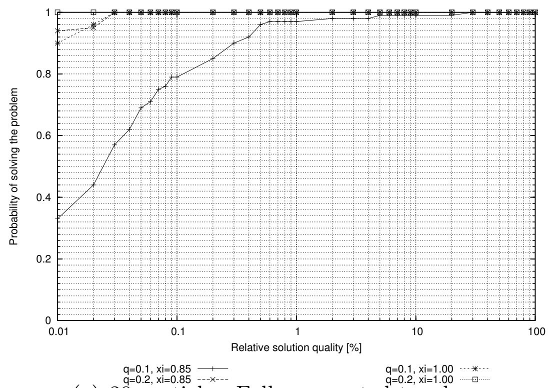


(e) 60 particles, Ring topology

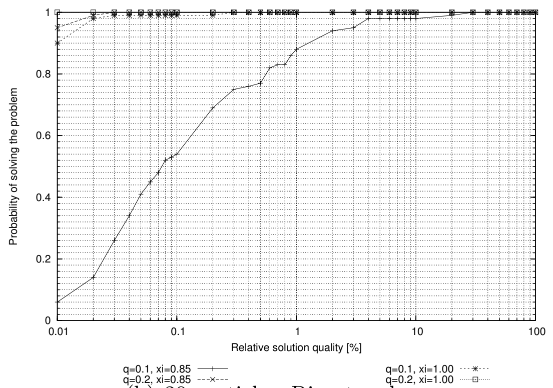


(f) 60 particles, Square topology

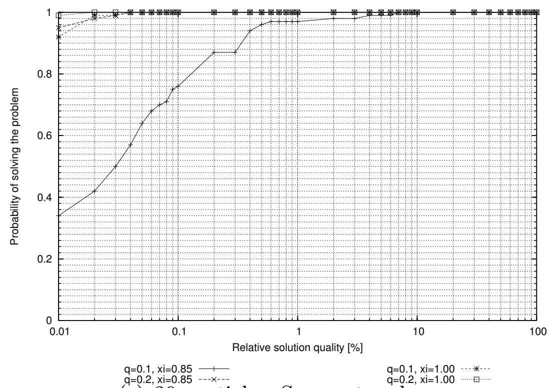
Figure 5.7: Run-Length Distributions on the Griewank Function. Estimation of Distribution Particle Swarm Optimizer. The solution quality demanded is of 0.001% away from the global optimum. Figures (a), (b), and (c) show the run-length distributions obtained with 20 particles. Figures (d), (e), and (f) show the run-length distributions obtained with 60 particles. Columns show the effect of using different population topologies.



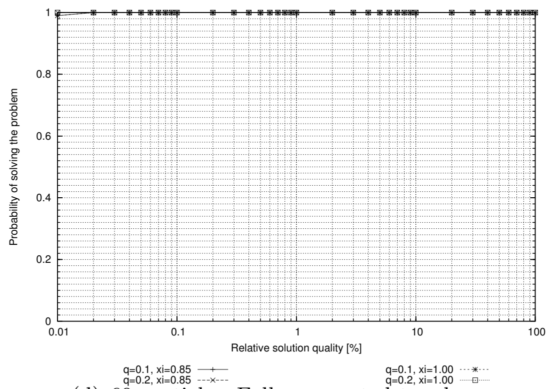
(a) 20 particles, Fully connected topology



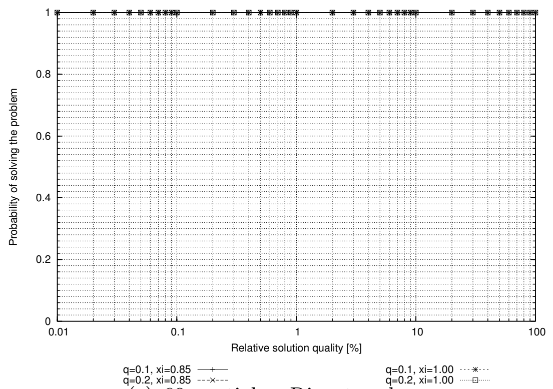
(b) 20 particles, Ring topology



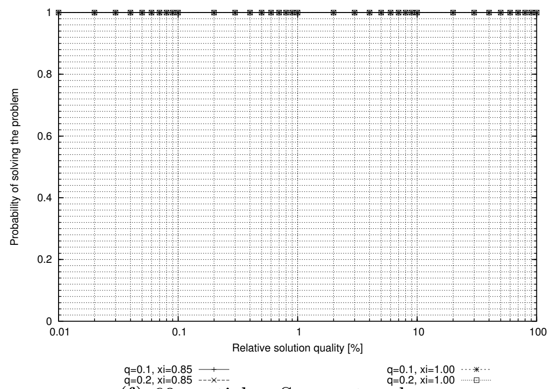
(c) 20 particles, Square topology



(d) 60 particles, Fully connected topology

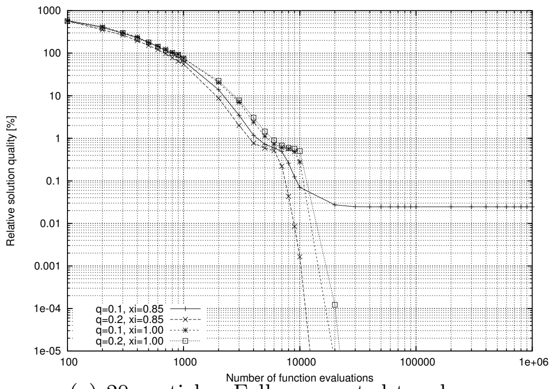


(e) 60 particles, Ring topology

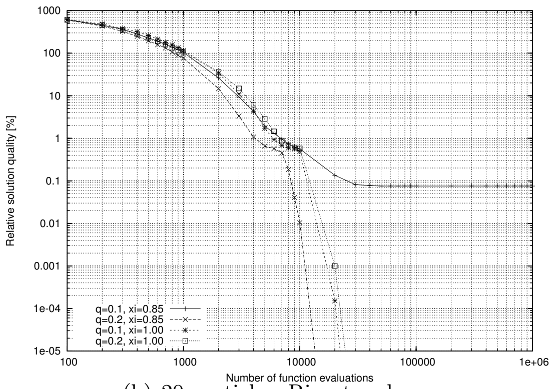


(f) 60 particles, Square topology

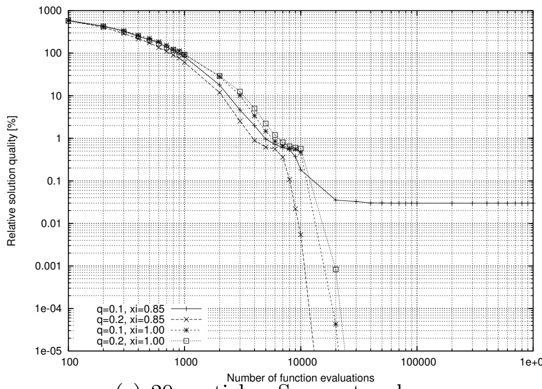
Figure 5.8: Solution-Quality Distributions on the Griewank Function. Estimation of Distribution Particle Swarm Optimizer. These results were obtained after 1 000 000 objective function evaluations. Figures (a), (b), and (c) show the solution-quality distributions obtained with 20 particles. Figures (d), (e), and (f) show the solution-quality distributions obtained with 60 particles. Columns show the effect of using different population topologies.



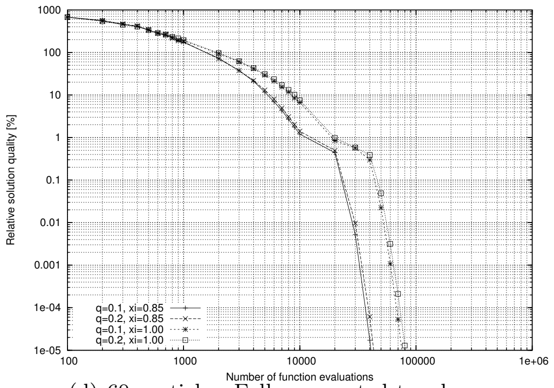
(a) 20 particles, Fully connected topology



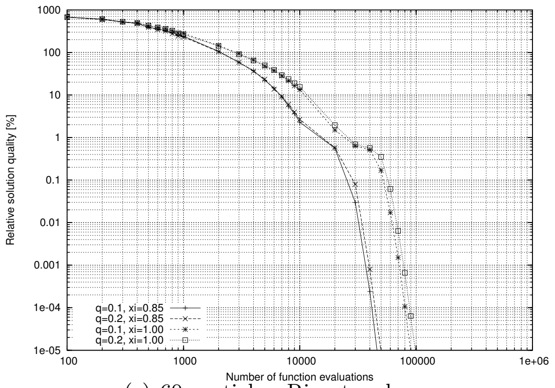
(b) 20 particles, Ring topology



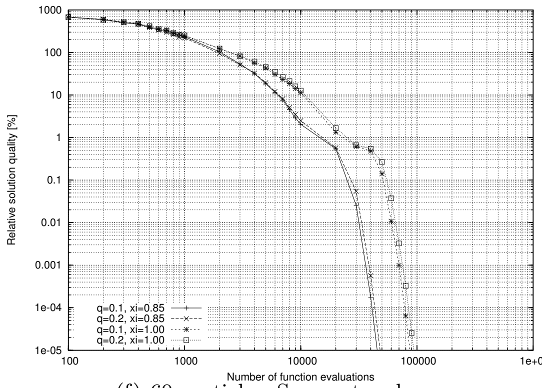
(c) 20 particles, Square topology



(d) 60 particles, Fully connected topology



(e) 60 particles, Ring topology



(f) 60 particles, Square topology

Figure 5.9: Solution quality development over time on the Griewank function. Estimation of Distribution Particle Swarm Optimizer. Figures (a), (b), and (c) show the solution quality development over time that is obtained with 20 particles. Figures (d), (e), and (f) show the solution quality development over time that is obtained obtained with 60 particles. Columns show the effect of using different population topologies. These results are based on medians.

Figure 5.11 shows the solution-quality distributions for the different EDPSO's configurations at 1 000 000 function evaluations. With 20 particles, the two best performances are obtained by the configurations in which $q = 0.2$. No significant difference can be appreciated between the results obtained with different population topologies. With 60 particles, the best performing configuration is the one with $\xi = 0.85$ and $q = 0.1$. This configuration shows a bimodal solution-quality *density* function. The other configurations abruptly drop at a solution quality level around 30% away from the optimum. In comparison with other optimizers, EDPSO exhibits a very similar behavior.

Figure 5.12 shows the median solution quality development over time achieved by the different EDPSO configurations. With 20 particles and irrespective of the population topology used, the fastest configuration is the one with $\xi = 0.85$ and $q = 0.1$. However, the one that achieves the best solution qualities is the one with $\xi = 1.0$ and $q = 0.2$. With 60 particles, the greediest configuration achieves the best results Although its improvement begins after the first 100 000 function evaluations.

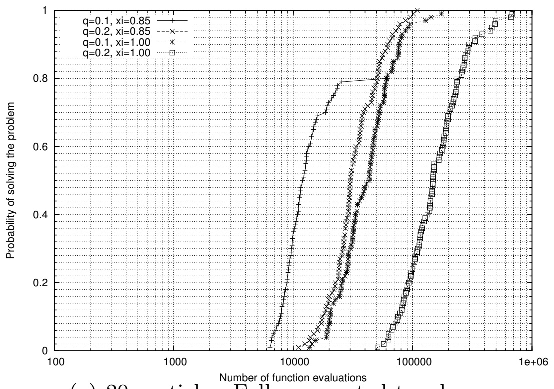
Results on the Rosenbrock Function

Figure 5.13 shows the run-length distributions obtained by the different EDPSO's configurations for a solution quality of 10% away from the optimum. The effect of an increase in the number of particles is much stronger than the effect of a change in the population topology. In all cases, the distributions change shape dramatically. With 20 particles, the distributions increase very rapidly between 10 000 and 100 000 function evaluations, but slowly afterwards (recall that the x-axis is in logarithmic scale). With 60 particles, the underlying *density* distributions become bimodal.

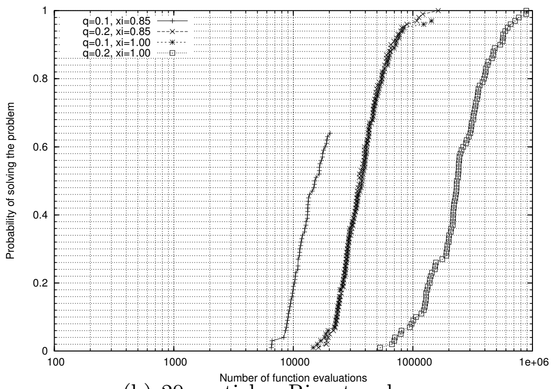
Compared with the performance of other particle swarm optimizers (cf. results published online), the performance of EDPSO in this function is not among the best ones. Irrespective of the population topology, it is dominated by all algorithms except by the decreasing inertia weight optimizer with the slowest inertia weight schedule, and HPSOTVAC.

Figure 5.14 shows the solution-quality distributions obtained by the different EDPSO's configurations after 1 000 000 function evaluations. The more explorative configuration, that is, the one with $\xi = 1.0$ and $q = 0.2$ is the one that reaches high quality solutions with higher probability than the others. The greediest configuration is the one with the worst performance. With 60 particles, the overall ability of the algorithm to find high quality solutions is hindered.

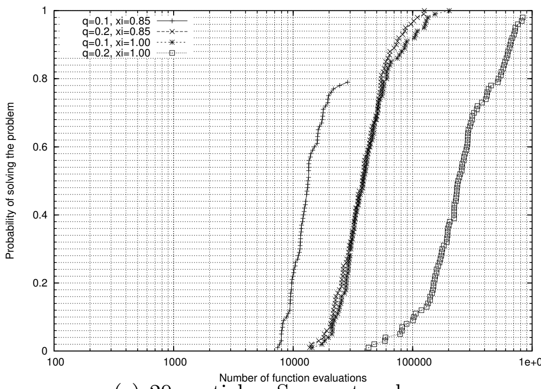
Figure 5.15 shows the solution quality development over time graphs for the different EDPSO's configurations. The fastest results are obtained with 20 particles. From these, up to 10 000 function evaluations, the best performing configuration is the one with $\xi = 0.85$ and $q = 0.2$. After that, and up to the end, the fastest configuration becomes the one with $\xi = 1.0$ and $q = 0.2$.



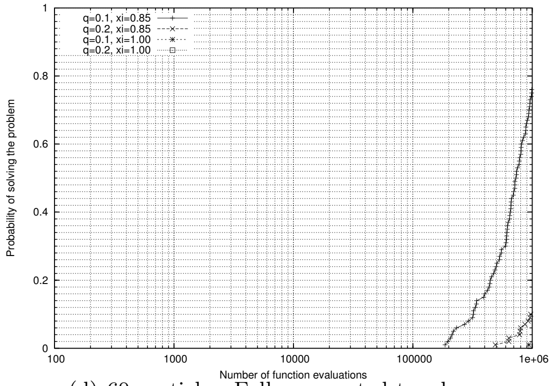
(a) 20 particles, Fully connected topology



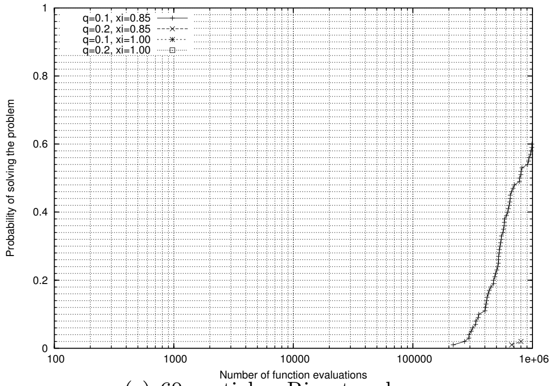
(b) 20 particles, Ring topology



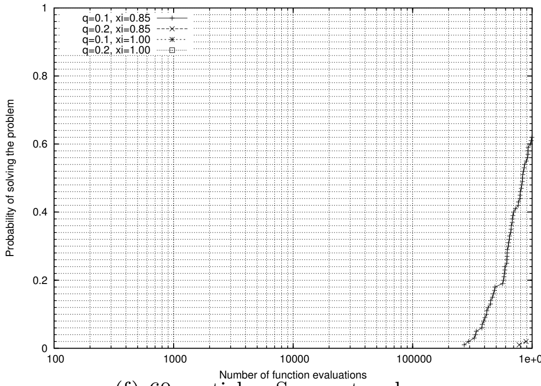
(c) 20 particles, Square topology



(d) 60 particles, Fully connected topology

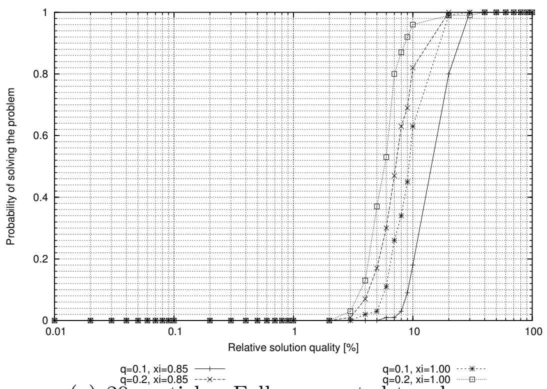


(e) 60 particles, Ring topology

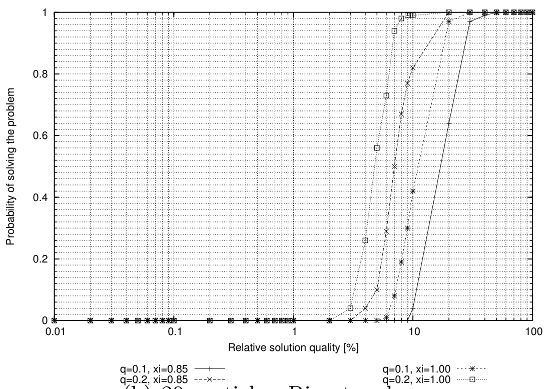


(f) 60 particles, Square topology

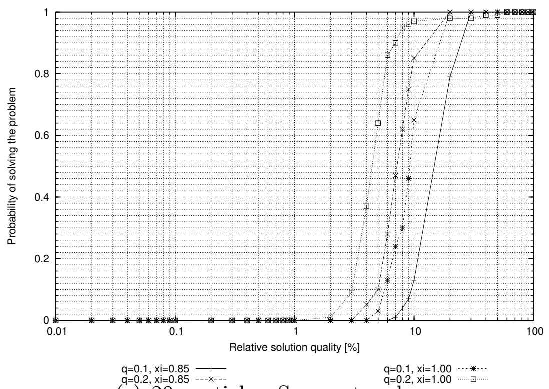
Figure 5.10: Run-Length Distributions on the Rastrigin Function. Estimation of Distribution Particle Swarm Optimizer. The solution quality demanded is of 20.0% away from the global optimum. Figures (a), (b), and (c) show the run-length distributions obtained with 20 particles. Figures (d), (e), and (f) show the run-length distributions obtained with 60 particles. Columns show the effect of using different population topologies.



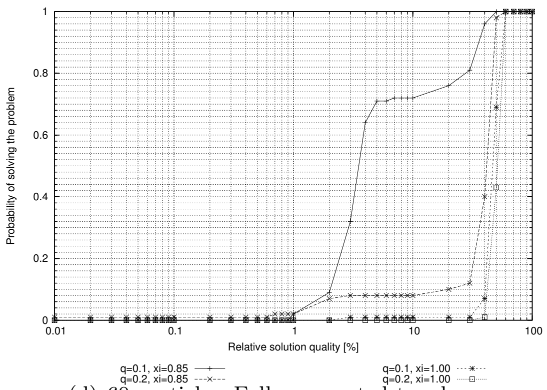
(a) 20 particles, Fully connected topology



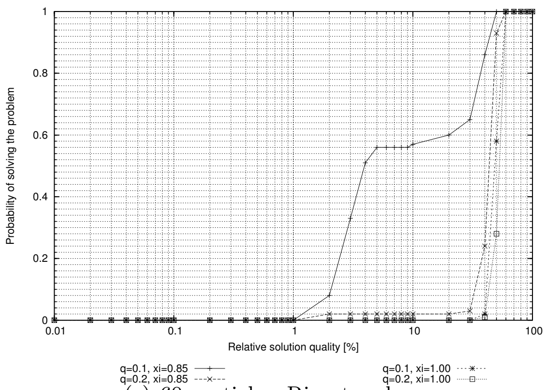
(b) 20 particles, Ring topology



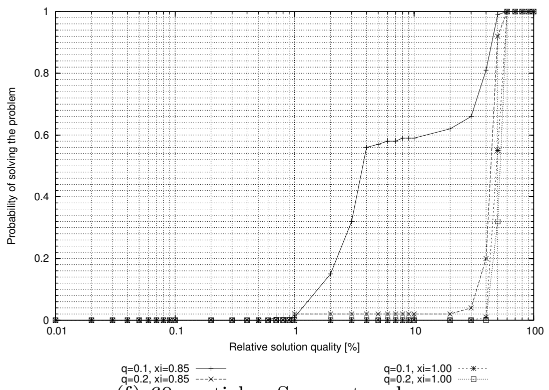
(c) 20 particles, Square topology



(d) 60 particles, Fully connected topology

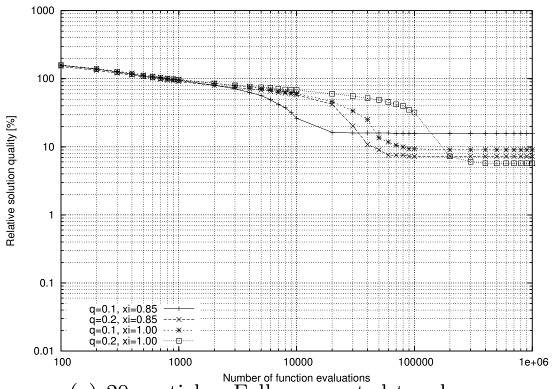


(e) 60 particles, Ring topology

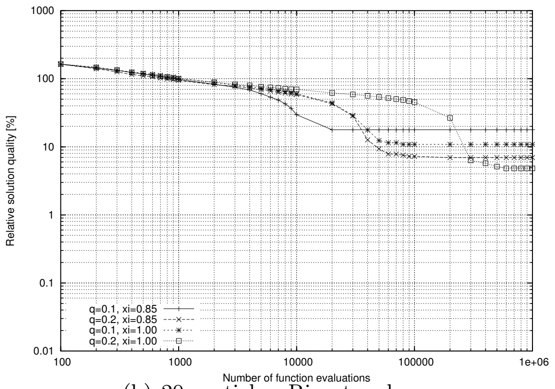


(f) 60 particles, Square topology

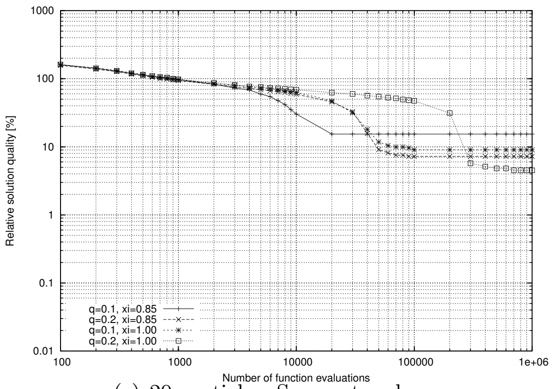
Figure 5.11: Solution-Quality Distributions on the Rastrigin Function. Estimation of Distribution Particle Swarm Optimizer. These results were obtained after 1 000 000 objective function evaluations. Figures (a), (b), and (c) show the solution-quality distributions obtained with 20 particles. Figures (d), (e), and (f) show the solution-quality distributions obtained with 60 particles. Columns show the effect of using different population topologies.



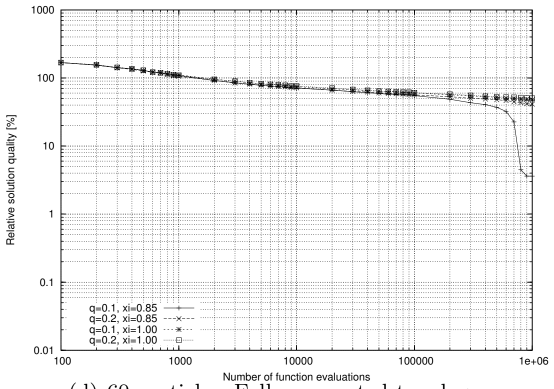
(a) 20 particles, Fully connected topology



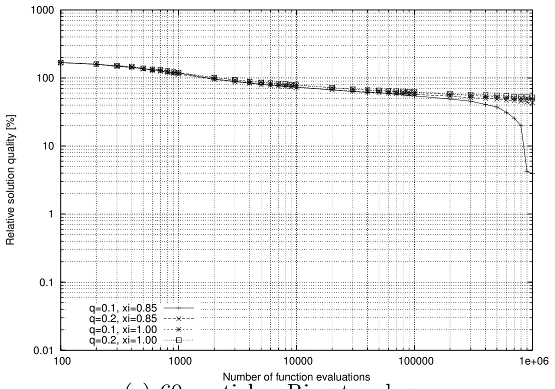
(b) 20 particles, Ring topology



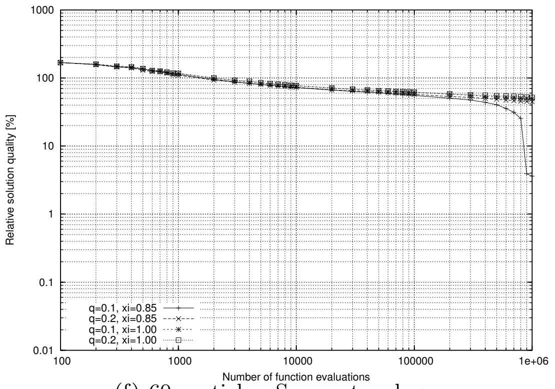
(c) 20 particles, Square topology



(d) 60 particles, Fully connected topology

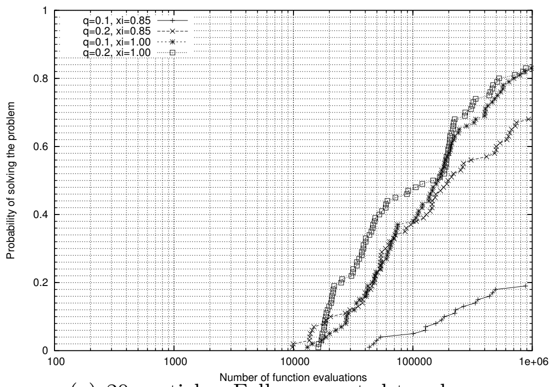


(e) 60 particles, Ring topology

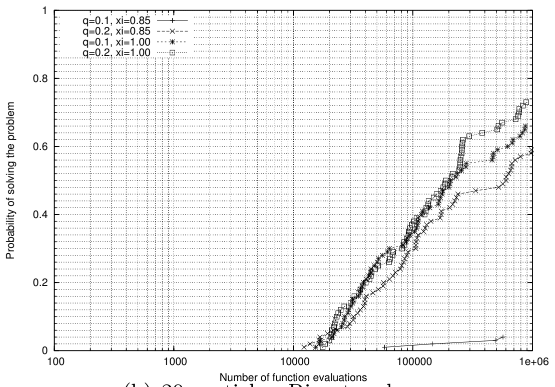


(f) 60 particles, Square topology

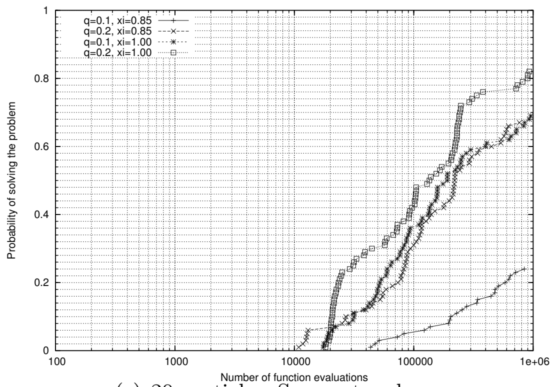
Figure 5.12: Solution quality development over time on the Rastrigin function. Estimation of Distribution Particle Swarm Optimizer. Figures (a), (b), and (c) show the solution quality development over time that is obtained with 20 particles. Figures (d), (e), and (f) show the solution quality development over time that is obtained obtained with 60 particles. Columns show the effect of using different population topologies. These results are based on medians.



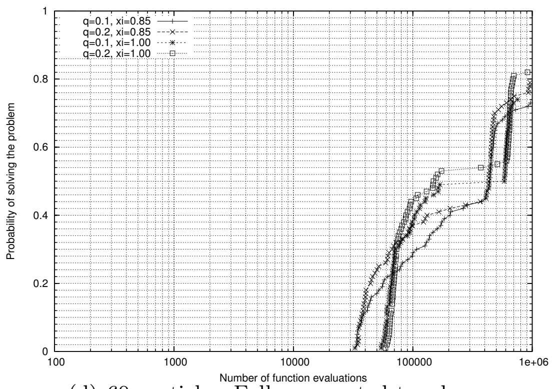
(a) 20 particles, Fully connected topology



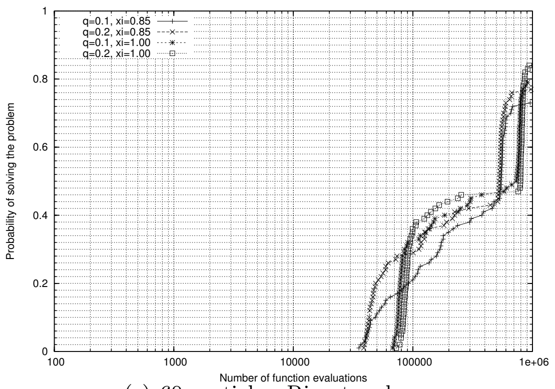
(b) 20 particles, Ring topology



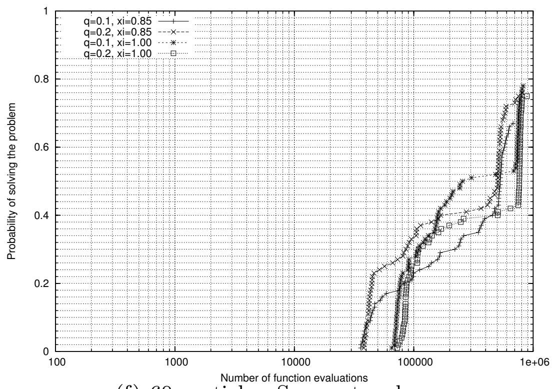
(c) 20 particles, Square topology



(d) 60 particles, Fully connected topology

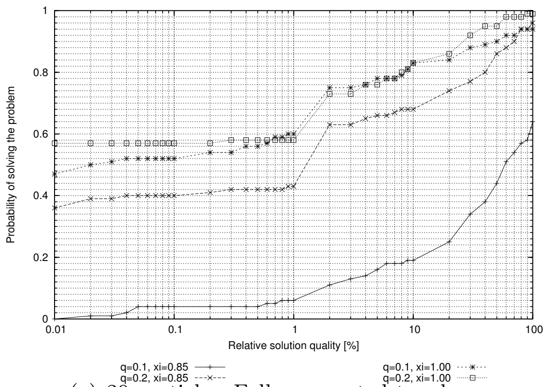


(e) 60 particles, Ring topology

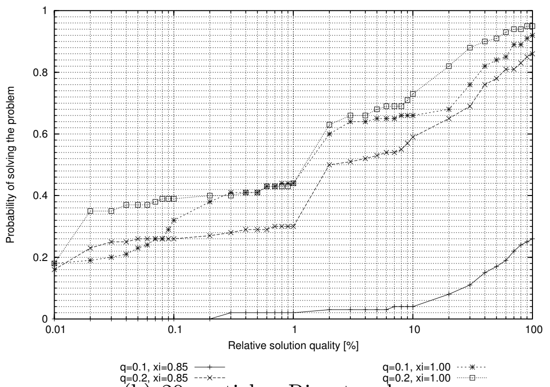


(f) 60 particles, Square topology

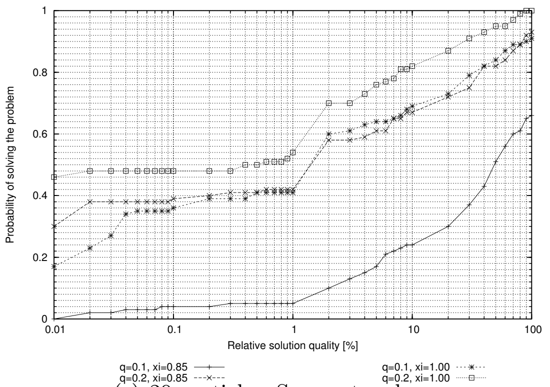
Figure 5.13: Run-Length Distributions on the Rosenbrock Function. Estimation of Distribution Particle Swarm Optimizer. The solution quality demanded is of 10.0% away from the global optimum. Figures (a), (b), and (c) show the run-length distributions obtained with 20 particles. Figures (d), (e), and (f) show the run-length distributions obtained with 60 particles. Columns show the effect of using different population topologies.



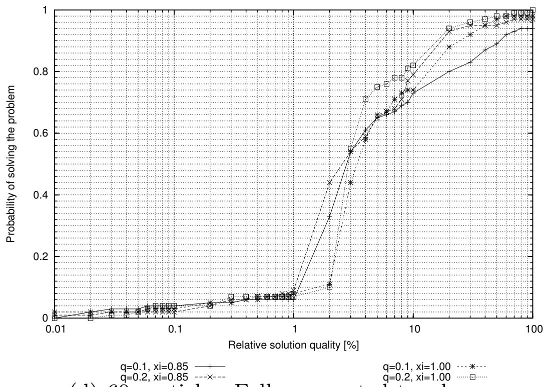
(a) 20 particles, Fully connected topology



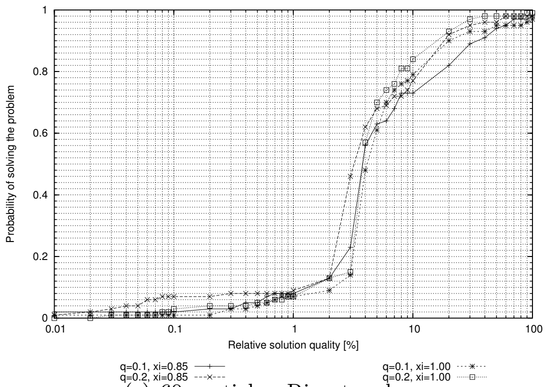
(b) 20 particles, Ring topology



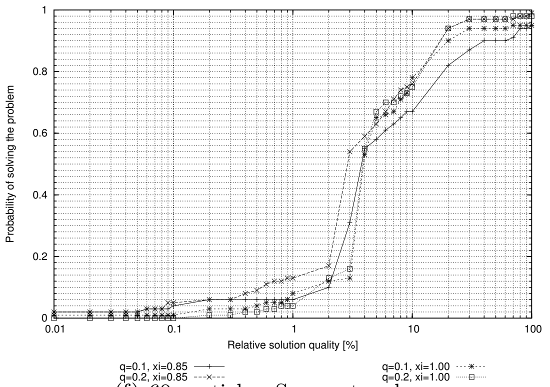
(c) 20 particles, Square topology



(d) 60 particles, Fully connected topology

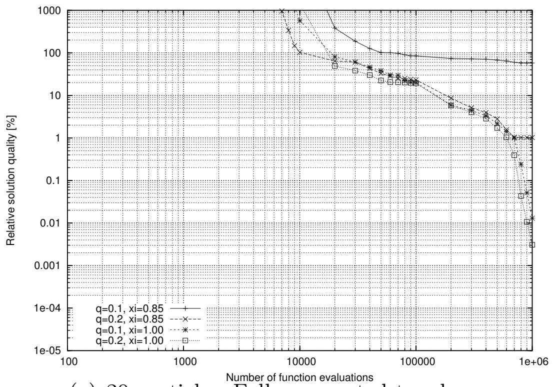


(e) 60 particles, Ring topology

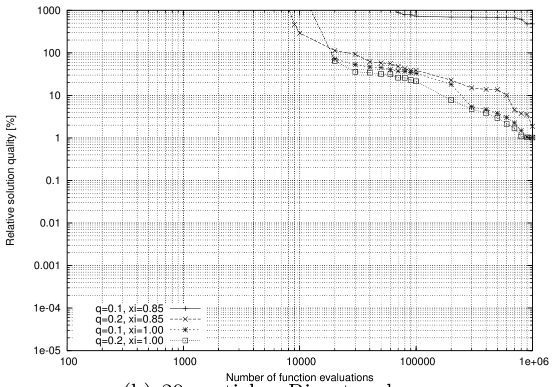


(f) 60 particles, Square topology

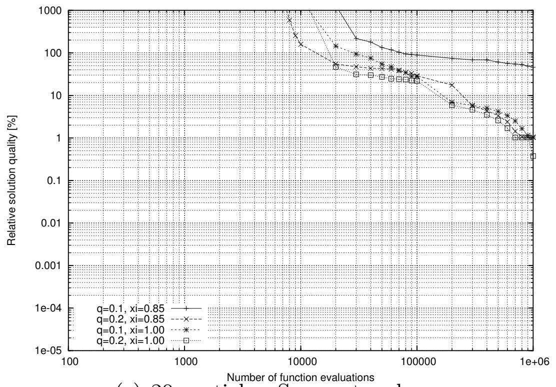
Figure 5.14: Solution-Quality Distributions on the Rosenbrock Function. Estimation of Distribution Particle Swarm Optimizer. These results were obtained after 1 000 000 objective function evaluations. Figures (a), (b), and (c) show the solution-quality distributions obtained with 20 particles. Figures (d), (e), and (f) show the solution-quality distributions obtained with 60 particles. Columns show the effect of using different population topologies.



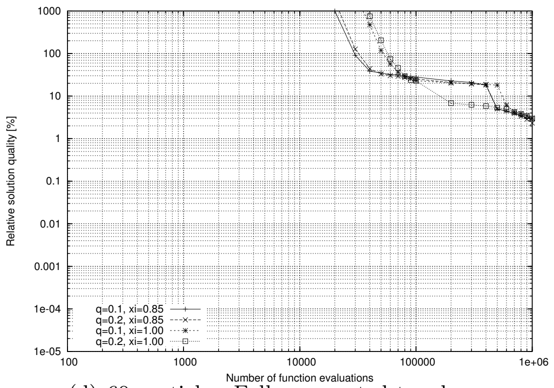
(a) 20 particles, Fully connected topology



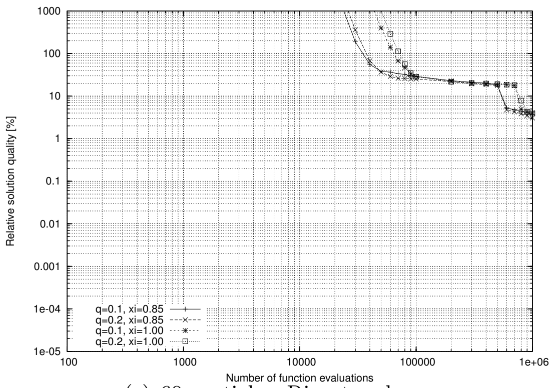
(b) 20 particles, Ring topology



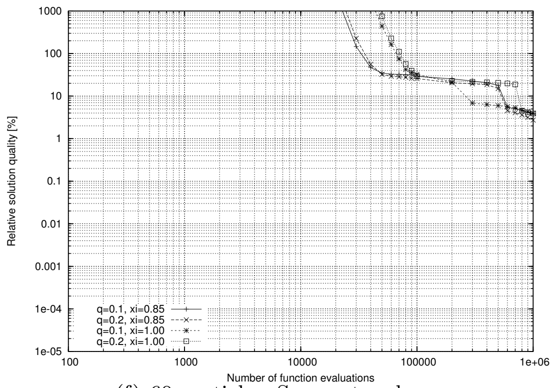
(c) 20 particles, Square topology



(d) 60 particles, Fully connected topology



(e) 60 particles, Ring topology



(f) 60 particles, Square topology

Figure 5.15: Solution quality development over time on the Rosenbrock function. Estimation of Distribution Particle Swarm Optimizer. Figures (a), (b), and (c) show the solution quality development over time that is obtained with 20 particles. Figures (d), (e), and (f) show the solution quality development over time that is obtained obtained with 60 particles. Columns show the effect of using different population topologies. These results are based on medians.

The idea of containing the dispersion of particles in the search space began soon after the introduction of the technique. In fact, the version that we now call "canonical" was devised as a way of controlling the "explosion" of the swarm which happens when the particles' velocity grows very rapidly and without limit. The decreasing inertia weight is another way for dealing with this problem. The effect, as we have seen, is that it forces the particles to reduce their speed.

The Estimation of Distribution Particle Swarm Optimizer (EDPSO) that we presented in this section can be thought of as another alternative for dealing with the same problem. EDPSO exploits the search history of the particle swarm to guide it to the most promising regions in the search space. The particle swarm search history is encoded in the set of vectors pointing to the best positions the particles have ever been in. The core idea is to contain the search within the best regions discovered by the algorithm. This containment is done in a probabilistic way so as not to hinder the exploration capabilities of the swarm. The parameters q and ξ help in balancing the exploration/exploitation behavior of the algorithm.

We have seen how EDPSO, with the appropriate parameter settings, performs better than the underlying canonical particle swam optimizer. These results, together with the the ones we obtained in Chapter 4, suggest that containing or forcing convergence is a good strategy if one wants to get good quality solutions fast. On the other hand, these greedy algorithms often stagnate in medium-long and long runs. Explorative configurations usually get better results in these cases.

5.4 Summary

In this chapter, we presented two improved particle swarm optimizers. One of them is not really an algorithmic variation but a strategy to control the way the decreasing inertia weight optimizer is run. The other algorithm is a hybrid approach in which a canonical particle swarm optimizer is equipped with an estimation of distribution mechanism that probabilistically contains the movement of the particles in the search space.

The main points presented in Section 5.1 are:

1. The analysis of run-length and solution-quality distributions can suggest ways of improving the performance of the studied algorithms.

2. Restarting a stochastic optimization algorithm several times, forces it to exhibit a geometric run-time distribution.

   Related points:

   - Stochastic optimization algorithms with an exponential run-time distribution is not affected by the use of restarts.
   - Algorithms whose run-time distributions grow slower than an exponential distribution improve their performance with the use of restarts.
   - The performance of algorithms with steeper-than-exponential run-time distributions are negatively affected by the use of restarts.

3. Using an adaptive restart strategy in the decreasing inertia weight optimizer makes the greatest difference when using an aggressive inertia weight schedule.

   Related points:

   - The faster the schedule, the earlier the divergence of the run-time distributions.

4. The effects of adaptive restart strategies are problem-dependent. What works fine for some problems, might not work at all for others.

   Related points:

   - In some problems, the adaptive restart mechanism is triggered prematurely. Depending on the specific behavior of the underlying algorithm on that problem, this can result in a performance deterioration or in a performance slow down.

The main points presented in Section 5.3 are:

1. The particle swarm search history (encoded in the set of vectors pointing to the positions in which the particles have obtained the best objective function evaluations) can be exploited in an alternative way to contain the movement of the particles in the most promising regions in the search space.

   Related points:

   - EDPSO is a hybrid approach that borrows some ideas from recent development in Ant Colony Optimization. It operates in the same way as a canonical particle swarm optimizer whenever the estimation of distribution mechanism does not contains the movement of particles. When a particle tries to move too far from a good region, EDPSO substitutes the move with a sample from a probabilistic model.
   - EDPSO is a promising algorithmic variant that performs better than other particle swarm variants.

Chapter 6

Conclusions

Particle Swarm Optimization [11, 23] is an 11 year-old field that has enjoyed an increasing interest from the industrial and academic worlds. Many algorithmic variants have been proposed and many particle swarm optimization algorithms have been used with success in many applications [24].

In spite of this success, there is no general agreement on which variant(s) can be considered as the state-of-the-art in the field. This has been the reason why most authors compare their variants with either, the original version, or the canonical particle swarm optimizer. In most cases, the variant is reported to "outperform" the algorithm of reference. As a result, the concept of a state-of-the-art particle swarm optimizer is a vague one.

In this document we reported a comparison among some of the most widely used particle swarm optimization algorithms. We embarked on this task to fill this missing aspect in the literature, and as a starting point for further developments.

The methodology we used for evaluating and comparing the different algorithms, helped us also to devise some improved versions.

In the following sections, we summarize the main points presented and discussed in each chapter of this document. We conclude this document pointing out some future research work.

6.1 An Empirical Evaluation Methodology of Stochastic Optimization Algorithms

The empirical methodology that was used in the work presented in this document is the one proposed by Hoos and Stützle [17]. It is based on the estimation of the run-length and solution-quality distributions exhibited by stochastic optimization algorithms when solving specific problems.

The run-time distribution is the cumulative distribution function of the random variable describing the time needed by a stochastic optimization algorithm to find a solution of a specific quality. Likewise, the solution-quality distribution is the cumulative distribution function of the random variable that represents the solution quality achieved by an algorithm exactly after some computational effort limit.

Run-time and solution-quality distributions provide information about the behavior of stochastic optimization algorithms that is useful for making comparisons and for devising improvements. Specifically, they can tell us how prone an algorithm is to stagnating. An algorithm with propensity to stagnate will show a slowly-increasing or a non-increasing probability (towards the right tail of the distribution) of finding a solution of a given quality over the allocated time limit.

The estimation of run-time and solution quality distributions does not impose a major burden in the process of collecting data. All we need to do is to run the algorithm several times and record information relative to solution improvement. In every run, we need to record the time or the number of critical operations, and the solution quality whenever a new best solution is found by the algorithm.

6.2 Empirical Evaluation of Particle Swarm Optimizers

The comparison reported in Chapter 4 was carried out using the most commonly used parameterization of the compared algorithms. This had the intention of comparing the performance of the different algorithms under the hypothetical scenario in which no *a priori* knowledge about the structure of the problem is available. This is the situation that, normally, one finds in a real-world application.

The results show that no algorithm dominates the rest. All algorithms perform better than another one in at least one of the ten problems in our benchmark set. However, some facts deserve mention:

- Depending on the problem and the demanded solution quality, different algorithms exhibit a stagnating behavior with different degrees of severity. This stagnating behavior can be alleviated by incrementing the population size or changing the population topology to a loosely connected one.

- Different algorithms are sensitive to a change in the population topology in different degrees. The sensitiveness also depends on the moment in time it is measured.

- In situations where the number of function evaluations must be maintained to the minimum, particle swarm optimizers with convergent properties (with highly connected topologies or low inertia weights) get the best results. For medium-long and long runs, algorithms with explorative properties (those with medianly or loosely connected topologies, or large inertia weights) perform best.

- In the case of the decreasing inertia weight particle swarm optimizer, faster inertia weight schedules provoke faster decelerations and spatial contractions of the particle swarm. Even though aggressive inertia weight schedules yield the best results for short runs. Faster inertia weight schedules provoke the algorithm to exhibit a stronger stagnating behavior. For long runs, slow schedules provide the best results. In general, it is a tradeoff between speed and quality.

6.3 Improved Particle Swarm Optimizers

After analyzing the behavior of the compared particle swarm optimizers, it was clear that most of them show a strong stagnating behavior. One way of alleviating this phenomenon is to increase the exploration capabilities of the algorithms. One of the easiest ways to deal with stagnation is to restart the algorithm once we have evidence that it cannot improve anymore the solution quality found so far. This mechanism was added to the decreasing inertia weight particle swarm optimizer, resulting in a significant improvement of its performance, specially when it used aggressive inertia weight schedules. The major problem of adaptive restart strategies is that the obtained improvement is problem-dependent.

The improvement achieved though the application of restarts can be explained if we realize that restarting a stochastic optimization algorithm several times, forces it to exhibit a geometric run-time distribution. If the geometric distribution thus obtained, grows faster than the algorithm's original run-time distribution, then we get an improvement. The significance of the improvement depends, obviously, on the departure of the algorithm's run-time distribution from an exponential one.

The second improved algorithm was designed borrowing some of ideas of recent developments in Ant Colony Optimization (ACO) [9], which is the other most prominent exponent of Swarm Intelligence [5], besides Particle Swarm Optimization.

The idea behind this variant is to exploit the search history of the particle swarm to guide it to the most promising regions in the search space. The Estimation of Distribution Particle Swarm Optimizer (EDPSO), as we called It, operates in the same way as a canonical particle swarm optimizer whenever the estimation of distribution mechanism (the feature taken from ACO) does not contains the movement of particles. When a particle tries to move too far from a good region, EDPSO substitutes the move with a sample from a probabilistic model. In the evaluation it was found that EDPSO is a promising algorithmic variant that performs better than other particle swarm variants.

6.4 Future Work

In this section, we mention some issues that we believe should be addressed to have a complete picture of the true performance of different particle swam optimizers. We first list the issues that would complement the results presented in this document, and then we list some potential research issues.

6.4.1 Complementary Work

In Chapter 3, we said that for comparing two or more stochastic optimization algorithms, one should take into account the specific aspects that are likely to be present in a practical application scenario. One such aspect is the dimensionality of the problem. Eight of the benchmark problems we used were 30-dimensional, and two were bidimensional. Aside from the bidimimensional problems (which have this dimensionality by definition), we decided to use 30-dimensional problems because we wanted to have results comparable to those reported in the literature. However, an evaluation on higher dimensions is the immediate next step to see whether the relative performance of the compared algorithms depends on the dimensionality of the problems and, if so, to which extent.

An issue that we did not discuss quantitatively in the document, is the interaction between the population topology and population size. We saw in Chapter 4 that there is indeed an interaction. The effects of a change in the population topology are not the same with 20 particles than with 60 particles. Establishing this relationship is another issue that remains to be done.

6.4.2 Research issues

Finding the optimal parameter set for a particle swarm optimizer solving a particular problem has been the interest of many researchers since the very birth of the technique. There have been analytical approaches (e.g. [7, 56]), trial and error approaches (e.g. [49, 58]) and heuristic ones (e.g. [30, 39, 40]), with varying degrees of success. Nevertheless, we believe that a fourth strategy based on Design of Experiments techniques, that have been used in industrial process tuning, offers the advantage of being based on solid statistics and that tackle the problem of a very reduced budget of possible experiments in an explicit way. This approach is beginning to be explored by Bartz-Beielstein [4].

Appendices

Appendix A

Parameterization of the Benchmark Functions Used in our Experiments

All the benchmark functions that we used in the study presented in this document have their global optima displaced and biased. In most cases, we used exactly the same values that were proposed in the set of benchmark functions used for the special session on real parameter optimization of the IEEE Congress of Evolutionary Computation 2005 [55].

Given a displacement vector $\vec{d}$ and a bias value b, all functions are transformed using

$$f'(\vec{x}) = f\left(\vec{x} - \vec{d}\right) + b\,, \tag{A.1}$$

where f is the original benchmark function, f' is the transformed benchmark function that has its global optimum in $\vec{o} + \vec{d}$, and has a function value of $o + b$. Vector $\vec{o}$ is the original location of the global optimum and o is its original function value, that is, $o = f(\vec{o})$.

Tables A.1 and A.2 show the displacement vectors and bias values used in our experiments. In our experiments, all functions except for two of them, are 30-dimensional. Easom and Schaffer functions are two-dimensional.

In order to compute the absolute value that corresponds to a relative solution quality specification (for minimization problems), we use

$$a = o + sign(o) \cdot (o \cdot r)\,, \tag{A.2}$$

where a is the absolute solution quality value, o is the objective function value of the known optimum, and r is the relative solution quality value. For example, suppose the objective function value of some problem's optimum is 100, and the relative solution quality is 0.1%. The solution quality in absolute values would be $100 + (100 \cdot 0.001) = 100.1$. On the other hand, if the optimum had a value of -100, the absolute value of a solution 0.1% away from it, would be -99.9.

Table A.1: Benchmark functions' displacement vectors and biases. Part I.

	Ackley	Easom	Griewank	Rastrigin	Rosenbrock
x_1	-1.68230e+1	0.0	-2.762684e+2	1.90050e+0	8.10232e+1
x_2	1.49769e+1	0.0	-1.191100e+1	-1.56440e+0	-4.83950e+1
x_3	6.16900e+0	N/A	-5.787884e+2	-9.78800e-1	1.92316e+1
x_4	9.55660e+0	N/A	-2.876486e+2	-2.25360e+0	-2.52310e+0
x_5	1.95417e+1	N/A	-8.438580e+1	2.49900e+0	7.04338e+1
x_6	-1.71900e+1	N/A	-2.286753e+2	-3.28530e+0	4.71774e+1
x_7	-1.88248e+1	N/A	-4.581516e+2	9.75900e-1	-7.83580e+0
x_8	8.51100e-1	N/A	-2.022145e+2	-3.66610e+0	-8.66693e+1
x_9	-1.51162e+1	N/A	-1.058642e+2	9.85000e-2	5.78532e+1
x_{10}	1.07934e+1	N/A	-9.648980e+1	-3.24650e+0	-9.95330e+0
x_{11}	7.40910e+0	N/A	-3.957468e+2	3.80600e+0	2.07778e+1
x_{12}	8.61710e+0	N/A	-5.729498e+2	-2.68340e+0	5.25486e+1
x_{13}	-1.65641e+1	N/A	-2.703641e+2	-1.37010e+0	7.59263e+1
x_{14}	-6.68000e+0	N/A	-5.668543e+2	4.18210e+0	4.28773e+1
x_{15}	1.45433e+1	N/A	-1.524204e+2	2.48560e+0	-5.82720e+1
x_{16}	7.04540e+0	N/A	-5.883819e+2	-4.22370e+0	-1.69728e+1
x_{17}	-1.86215e+1	N/A	-2.828892e+2	3.36530e+0	7.83845e+1
x_{18}	1.45561e+1	N/A	-4.888865e+2	2.15320e+0	7.50427e+1
x_{19}	-1.15942e+1	N/A	-3.469817e+2	-3.09290e+0	-1.61513e+1
x_{20}	-1.91531e+1	N/A	-4.530447e+2	4.31050e+0	7.08569e+1
x_{21}	-4.73720e+0	N/A	-5.065857e+2	-2.98610e+0	-7.95795e+1
x_{22}	9.25900e-1	N/A	-4.759987e+2	3.49360e+0	-2.64837e+1
x_{23}	1.32412e+1	N/A	-3.620492e+2	-2.72890e+0	5.63699e+1
x_{24}	-5.29470e+0	N/A	-2.332367e+2	-4.12660e+0	-8.82249e+1
x_{25}	1.84160e+0	N/A	-4.919864e+2	-2.59000e+0	-6.49996e+1
x_{26}	4.56180e+0	N/A	-5.440898e+2	1.31240e+0	-5.35022e+1
x_{27}	-1.88905e+1	N/A	-7.344560e+1	-1.79900e+0	-5.42300e+1
x_{28}	9.80080e+0	N/A	-5.269011e+2	-1.18900e+0	1.86826e+1
x_{29}	-1.54265e+1	N/A	-5.022561e+2	-1.05300e-1	-4.10061e+1
x_{30}	1.27220e+0	N/A	-5.372353e+2	-3.10740e+0	-5.42134e+1
bias	-140.0	0.0	-180.0	-330.0	390.0

Table A.2: Benchmark functions' displacement vectors and biases. Part II.

	Salomon	Schaffer	Schwefel	Sphere	Step
x_1	-1.68230e+1	-7.36029e+1	0.0	-3.93119e+1	0.0
x_2	0.00769e+0	-2.35497e+1	0.0	5.88999e+1	0.0
x_3	6.16900e+0	N/A	0.0	-4.63224e+1	0.0
x_4	9.55660e+0	N/A	0.0	-7.46515e+1	0.0
x_5	1.95417e+1	N/A	0.0	-1.67997e+1	0.0
x_6	-1.71900e+1	N/A	0.0	-8.05441e+1	0.0
x_7	-1.88248e+1	N/A	0.0	-1.05935e+1	0.0
x_8	8.51100e-1	N/A	0.0	2.49694e+1	0.0
x_9	-1.51162e+1	N/A	0.0	8.98384e+1	0.0
x_{10}	1.07934e+1	N/A	0.0	9.11190e+0	0.0
x_{11}	7.00000e+0	N/A	0.0	-1.07443e+1	0.0
x_{12}	8.61710e+0	N/A	0.0	-2.78558e+1	0.0
x_{13}	-1.65641e+1	N/A	0.0	-1.25806e+1	0.0
x_{14}	-6.68000e+0	N/A	0.0	7.59300e+0	0.0
x_{15}	1.45433e+1	N/A	0.0	7.48127e+1	0.0
x_{16}	7.04540e+0	N/A	0.0	6.84959e+1	0.0
x_{17}	-1.86215e+1	N/A	0.0	-5.34293e+1	0.0
x_{18}	1.45561e+1	N/A	0.0	7.88544e+1	0.0
x_{19}	-1.05942e+1	N/A	0.0	-6.85957e+1	0.0
x_{20}	-1.91531e+1	N/A	0.0	6.37432e+1	0.0
x_{21}	-4.73720e+0	N/A	0.0	3.13470e+1	0.0
x_{22}	9.25900e-1	N/A	0.0	-3.75016e+1	0.0
x_{23}	1.32412e+1	N/A	0.0	3.38929e+1	0.0
x_{24}	-5.29470e+1	N/A	0.0	-8.88045e+1	0.0
x_{25}	1.84160e+0	N/A	0.0	-7.87719e+1	0.0
x_{26}	4.56180e+0	N/A	0.0	-6.64944e+1	0.0
x_{27}	-1.88905e+1	N/A	0.0	4.41972e+1	0.0
x_{28}	9.80080e+0	N/A	0.0	1.83836e+1	0.0
x_{29}	-1.54265e+1	N/A	0.0	2.65212e+1	0.0
x_{30}	1.27220e+0	N/A	0.0	8.44723e+1	0.0
bias	-100.0	-300.0	100.0	-450.0	-200.0

Appendix B

Empirical Evaluation of Particle Swarm Optimizers: Median Solution Qualities

In this appendix we present the median solution qualities, after a certain number of function evaluations, achieved by the compared algorithms

Table B.1: Median solution qualities achieved by the compared particle swarm optimizers using 40 particles and a fully connected topology. Each cell contains the median solution quality achieved by the corresponding optimizer after 1 000, 10 000, 100 000 and 1 000 000 function evaluations.

Algorithm	High dimensional								Low dimensional	
	Multimodal						Unimodal		Multimodal	Unimodal
	Ackley	Step	Griewank	Rastrigin	Schwefel	Salomon	Sphere	Rosenbrock	Schaffer	Easom
Canonical	1.184e+01	5.300e+01	7.680e+01	9.964e+01	9.623e+03	1.127e+01	2.790e+03	5.011e+07	3.303e-03	7.835e+00
	1.318e+00	1.500e+01	2.043e-01	2.295e+01	4.140e+03	1.100e+00	5.657e-02	1.624e+02	6.764e-12	0.000e+00
	1.176e+00	2.500e+00	1.094e-02	2.171e+01	3.988e+03	4.999e-01	2.533e-14	4.566e+00	0.000e+00	0.000e+00
	1.176e+00	1.500e+00	1.094e-02	2.171e+01	3.988e+03	4.999e-01	2.533e-14	7.274e-06	0.000e+00	0.000e+00
Decreasing-IW	1.423e+01	6.450e+01	2.379e+02	1.209e+02	1.022e+04	2.065e+01	9.485e+03	1.088e+08	9.707e-03	9.662e+01
	1.313e+01	5.400e+01	1.198e+02	9.841e+01	9.781e+03	1.608e+01	5.161e+03	1.053e+07	3.256e-03	4.823e+00
	1.117e+01	3.700e+01	5.415e+01	7.784e+01	8.483e+03	1.165e+01	2.405e+03	1.990e+06	3.125e-04	7.470e-04
	6.071e-14	0.000e+00	8.207e-03	4.523e+00	1.303e+03	2.999e-01	1.267e-14	4.689e+00	0.000e+00	0.000e+00
Increasing-IW	1.094e+01	4.600e+01	6.135e+01	8.864e+01	8.716e+03	9.980e+00	2.282e+03	1.072e+08	3.243e-03	8.263e-02
	8.481e-01	1.450e+01	1.452e-01	1.878e+01	3.469e+03	1.401e+00	2.557e-02	3.718e+02	5.152e-06	0.000e+00
	8.143e-14	5.000e-01	5.476e-03	1.598e+01	3.178e+03	3.999e-01	2.533e-14	1.787e+01	0.000e+00	0.000e+00
	8.143e-14	0.000e+00	5.476e-03	1.206e+01	2.477e+03	3.999e-01	1.267e-14	3.483e+00	0.000e+00	0.000e+00
Stochastic-IW	1.209e+01	5.350e+01	9.421e+01	1.011e+02	9.688e+03	1.211e+01	3.426e+03	6.166e+07	3.291e-03	7.405e+00
	1.403e+00	1.650e+01	5.666e-01	2.432e+01	4.040e+03	1.411e+00	4.741e-01	3.344e+02	9.212e-11	0.000e+00
	6.652e-01	1.000e+00	6.842e-03	1.779e+01	3.336e+03	3.999e-01	1.267e-14	5.690e+00	0.000e+00	0.000e+00
	6.652e-01	5.000e-01	6.842e-03	1.779e+01	3.336e+03	3.999e-01	1.267e-14	2.226e-04	0.000e+00	0.000e+00
FIPS	7.611e+00	4.400e+01	7.675e+01	7.956e+01	1.014e+04	2.986e+00	1.839e+03	2.067e+09	3.246e-03	4.633e+00
	6.579e+00	4.400e+01	6.589e+01	5.180e+01	1.014e+04	2.805e+00	1.514e+03	1.939e+09	3.239e-03	0.000e+00
	6.579e+00	4.400e+01	6.589e+01	5.177e+01	1.006e+04	2.805e+00	1.514e+03	1.939e+09	3.239e-03	0.000e+00
	6.579e+00	4.400e+01	6.589e+01	5.177e+01	1.000e+04	2.805e+00	1.514e+03	1.939e+09	3.239e-03	0.000e+00
HPSOTVAC	1.254e+01	5.550e+01	2.381e+02	9.843e+01	9.366e+03	7.307e+00	7.165e+03	3.790e+09	5.306e-03	7.037e+01
	8.661e+00	3.000e+01	4.806e+00	4.620e+01	5.903e+03	4.644e+00	1.475e+02	1.652e+06	3.239e-03	1.000e-13
	5.278e-02	8.000e+00	7.877e-04	1.693e+01	1.935e+03	2.000e+00	1.309e-07	3.824e+01	0.000e+00	0.000e+00
	1.413e-07	1.000e+00	6.333e-14	7.490e-10	1.540e+03	5.999e-01	5.044e-14	1.021e-01	0.000e+00	0.000e+00
AHPSO	1.208e+01	5.250e+01	8.382e+01	9.835e+01	9.639e+03	7.331e+01	3.298e+03	8.997e+07	3.311e-03	7.178e+00
	1.306e+00	1.500e+01	3.685e-01	2.167e+01	4.073e+03	7.003e+01	1.182e-01	1.851e+02	7.053e-08	0.000e+00
	1.270e+00	6.000e+00	6.845e-03	1.990e+01	3.988e+03	6.985e+01	2.533e-14	4.985e+00	0.000e+00	0.000e+00
	1.270e+00	3.000e+00	6.845e-03	1.990e+01	3.988e+03	6.982e+01	1.267e-14	1.150e-03	0.000e+00	0.000e+00

Table B.2: Median solution qualities achieved by the compared particle swarm optimizers using 40 particles and a ring topology. Each cell contains the median solution quality achieved by the corresponding optimizer after 1 000, 10 000, 100 000 and 1 000 000 function evaluations.

Algorithm	High dimensional								Low dimensional	
	Multimodal						Unimodal		Multimodal	Unimodal
	Ackley	Step	Griewank	Rastrigin	Schwefel	Salomon	Sphere	Rosenbrock	Schaffer	Easom
Canonical	1.385e+01	6.000e+01	2.376e+02	1.124e+02	9.515e+03	1.594e+01	8.431e+03	1.331e+09	3.716e-03	4.121e+01
	3.292e+00	2.350e+01	1.273e+00	3.709e+01	5.526e+03	3.125e+00	2.664e+01	3.651e+04	1.255e-03	0.000e+00
	3.804e-11	5.000e+00	1.556e-14	1.990e+01	4.383e+03	2.999e-01	0.000e+00	6.865e+00	0.000e+00	0.000e+00
	2.000e-14	0.000e+00	0.000e+00	1.930e+01	4.382e+03	1.999e-01	0.000e+00	8.359e-02	0.000e+00	0.000e+00
Decreasing-IW	1.469e+01	6.500e+01	3.766e+02	1.391e+02	1.023e+04	2.262e+01	1.446e+04	1.336e+09	1.248e-02	1.000e+02
	1.367e+01	5.750e+01	1.394e+02	1.022e+02	9.719e+03	1.706e+01	6.158e+03	1.650e+07	3.243e-03	2.671e+00
	1.028e+01	3.300e+01	3.715e+01	6.753e+01	7.407e+03	9.708e+00	1.547e+03	1.033e+06	1.970e-04	3.286e-04
	2.000e-14	0.000e+00	1.556e-14	8.744e+00	1.559e+03	2.999e-01	0.000e+00	1.126e+00	0.000e+00	0.000e+00
Increasing-IW	1.336e+01	5.550e+01	1.788e+02	1.040e+02	9.025e+03	1.421e+01	6.400e+03	1.300e+09	3.344e-03	6.270e+00
	3.658e+00	2.300e+01	1.263e+00	3.487e+01	4.848e+03	4.100e+00	2.895e+01	4.440e+04	3.239e-03	0.000e+00
	9.892e-09	4.500e+00	1.556e-14	1.960e+01	3.811e+03	4.999e-01	1.267e-14	1.689e+01	0.000e+00	0.000e+00
	2.000e-14	5.000e-01	0.000e+00	1.393e+01	3.005e+03	2.999e-01	0.000e+00	1.732e+00	0.000e+00	0.000e+00
Stochastic-IW	1.389e+01	6.050e+01	2.448e+02	1.168e+02	9.519e+03	1.623e+01	8.950e+03	1.478e+09	3.855e-03	4.933e+01
	4.335e+00	2.650e+01	2.228e+00	4.059e+01	5.479e+03	3.817e+00	6.672e+01	8.165e+04	7.594e-04	0.000e+00
	2.737e-08	4.500e+00	6.333e-14	1.869e+01	4.047e+03	3.999e-01	1.267e-14	1.104e+01	0.000e+00	0.000e+00
	2.000e-14	0.000e+00	0.000e+00	1.719e+01	3.909e+03	2.999e-01	0.000e+00	8.481e-02	0.000e+00	0.000e+00
FIPS	1.291e+01	5.800e+01	1.595e+02	1.041e+02	9.779e+03	1.247e+01	5.627e+03	1.455e+09	4.908e-03	6.697e+01
	8.404e-01	1.700e+01	5.691e-01	5.188e+01	8.293e+03	1.443e+00	4.932e-01	4.395e+03	7.007e-04	1.000e-13
	2.000e-14	3.500e+00	4.739e-13	1.758e+01	4.524e+03	3.999e-01	0.000e+00	2.931e+01	0.000e+00	0.000e+00
	0.000e+00	5.000e-01	0.000e+00	6.397e+00	4.738e+02	2.999e-01	0.000e+00	9.379e+00	0.000e+00	0.000e+00
HPSOTVAC	1.371e+01	6.000e+01	3.016e+02	1.184e+02	9.228e+03	1.256e+01	9.425e+03	3.326e+09	1.245e-02	9.789e+01
	5.674e+00	2.900e+01	5.178e+00	6.235e+01	6.114e+03	4.605e+00	1.719e+02	9.291e+05	3.239e-03	1.505e-06
	1.750e-03	7.500e+00	3.170e-04	1.672e+01	2.608e+03	9.999e-01	1.727e-05	3.588e+01	0.000e+00	0.000e+00
	3.220e-07	0.000e+00	1.556e-14	8.942e-08	1.184e+03	3.999e-01	1.011e-13	2.762e+01	0.000e+00	0.000e+00
AHPSO	1.208e+01	5.250e+01	8.382e+01	9.835e+01	9.639e+03	7.331e+01	3.298e+03	8.997e+07	3.311e-03	7.178e+00
	1.306e+00	1.500e+01	3.685e-01	2.167e+01	4.073e+03	7.003e+01	1.182e-01	1.851e+02	7.053e-08	0.000e+00
	1.270e+00	6.000e+00	6.845e-03	1.990e+01	3.988e+03	6.985e+01	2.533e-14	4.985e+00	0.000e+00	0.000e+00
	1.270e+00	3.000e+00	6.845e-03	1.990e+01	3.988e+03	6.982e+01	1.267e-14	1.150e-03	0.000e+00	0.000e+00

Table B.3: Median solution qualities achieved by the compared particle swarm optimizers using 40 particles and a square topology. Each cell contains the median solution quality achieved by the corresponding optimizer after 1 000, 10 000, 100 000 and 1 000 000 function evaluations.

Algorithm	High dimensional								Low dimensional	
	Multimodal						Unimodal		Multimodal	Unimodal
	Ackley	Step	Griewank	Rastrigin	Schwefel	Salomon	Sphere	Rosenbrock	Schaffer	Easom
Canonical	1.302e+01	5.800e+01	1.459e+02	1.005e+02	9.443e+03	1.368e+01	5.441e+03	3.340e+08	3.476e-03	2.236e+01
	1.843e+00	1.900e+01	6.560e-01	2.509e+01	4.811e+03	1.911e+00	3.392e+00	1.702e+03	5.168e-07	0.000e+00
	4.071e-14	1.500e+00	1.556e-14	1.538e+01	3.889e+03	2.999e-01	1.267e-14	2.008e+01	0.000e+00	0.000e+00
	2.000e-14	0.000e+00	1.556e-14	1.477e+01	3.849e+03	1.999e-01	1.267e-14	8.171e-03	0.000e+00	0.000e+00
Decreasing-IW	1.454e+01	6.500e+01	2.933e+02	1.279e+02	1.021e+04	2.170e+01	1.198e+04	3.932e+08	1.273e-02	9.925e+01
	1.340e+01	5.600e+01	1.314e+02	9.969e+01	9.644e+03	1.680e+01	5.615e+03	1.262e+07	3.247e-03	3.899e+00
	1.048e+01	3.350e+01	3.870e+01	7.056e+01	7.581e+03	1.020e+01	1.688e+03	1.104e+06	2.524e-04	2.166e-04
	2.000e-14	0.000e+00	1.556e-14	6.332e+00	1.303e+03	2.999e-01	1.267e-14	2.957e+00	0.000e+00	0.000e+00
Increasing-IW	1.220e+01	5.300e+01	1.004e+02	9.054e+01	8.982e+03	1.207e+01	4.097e+03	3.083e+08	3.255e-03	2.278e+00
	2.125e+00	1.850e+01	6.552e-01	2.332e+01	4.265e+03	2.757e+00	3.464e+00	2.808e+03	7.362e-05	0.000e+00
	2.236e-13	2.500e+00	1.556e-14	1.387e+01	3.277e+03	2.999e-01	1.267e-14	1.774e+01	0.000e+00	0.000e+00
	2.000e-14	0.000e+00	1.556e-14	9.563e+00	2.291e+03	2.999e-01	1.267e-14	4.514e+00	0.000e+00	0.000e+00
Stochastic-IW	1.320e+01	5.800e+01	1.606e+02	1.034e+02	9.560e+03	1.414e+01	5.441e+03	4.052e+08	3.682e-03	2.683e+01
	2.617e+00	2.100e+01	9.239e-01	3.035e+01	5.008e+03	2.500e+00	3.392e+00	7.065e+03	1.447e-05	0.000e+00
	1.033e-11	2.000e+00	1.556e-14	1.357e+01	3.514e+03	2.999e-01	1.267e-14	1.918e+01	0.000e+00	0.000e+00
	2.000e-14	0.000e+00	1.556e-14	1.296e+01	3.356e+03	1.999e-01	1.267e-14	2.193e-01	0.000e+00	0.000e+00
FIPS	1.024e+01	4.900e+01	7.059e+01	8.660e+01	1.004e+04	6.886e+00	2.299e+03	8.459e+08	3.918e-03	5.830e+01
	1.073e+00	2.300e+01	2.462e-01	3.820e+01	9.871e+03	1.500e+00	2.065e-01	2.721e+07	3.174e-04	0.000e+00
	1.073e+00	2.200e+01	1.388e-01	1.516e+01	9.601e+03	1.500e+00	1.800e-01	2.707e+07	3.250e-10	0.000e+00
	1.073e+00	2.150e+01	1.177e-01	1.429e+01	5.801e+03	1.500e+00	1.800e-01	2.707e+07	0.000e+00	0.000e+00
HPSOTVAC	1.286e+01	5.800e+01	2.495e+02	1.076e+02	9.311e+03	8.513e+00	7.504e+03	3.177e+09	5.434e-03	9.941e+01
	4.363e+00	2.450e+01	2.831e+00	5.466e+01	6.264e+03	3.104e+00	8.685e+01	3.967e+05	3.239e-03	1.690e-09
	5.899e-04	7.500e+00	2.040e-05	1.532e+01	2.311e+03	1.100e+00	1.713e-06	4.112e+01	0.000e+00	0.000e+00
	1.269e-07	5.000e-01	1.556e-14	2.385e-09	1.303e+03	5.999e-01	2.533e-14	1.857e+01	0.000e+00	0.000e+00
AHPSO	1.208e+01	5.250e+01	8.382e+01	9.835e+01	9.639e+03	7.331e+01	3.298e+03	8.997e+07	3.311e-03	7.178e+00
	1.306e+00	1.500e+01	3.685e-01	2.167e+01	4.073e+03	7.003e+01	1.182e-01	1.851e+02	7.053e-08	0.000e+00
	1.270e+00	6.000e+00	6.845e-03	1.990e+01	3.988e+03	6.985e+01	2.533e-14	4.985e+00	0.000e+00	0.000e+00
	1.270e+00	3.000e+00	6.845e-03	1.990e+01	3.988e+03	6.982e+01	1.267e-14	1.150e-03	0.000e+00	0.000e+00

Appendix C

Different Inertia Weight Schedules in the Time-Decreasing Inertia Weight Particle Swarm Optimizer: Median Solution Qualities

In this appendix we present the median solution qualities, after a certain number of function evaluations, achieved by the compared configurations.

Table C.1: [Median solution qualities achieved by the decreasing inertia weight optimizers with different schedules based on median solution qualities and a fully connected topology. Each cell contains the median solution quality achieved by the corresponding optimizer after 1 000, 10 000, 100 000 and 1 000 000 function evaluations.

Schedule	High dimensional								Low dimensional	
	Multimodal						Unimodal		Multimodal	Unimodal
	Ackley	Step	Griewank	Rastrigin	Schwefel	Salomon	Sphere	Rosenbrock	Schaffer	Easom
100	1.085e+01	4.800e+01	5.733e+01	8.795e+01	8.800e+03	1.003e+01	2.235e+03	9.209e+07	3.245e-03	1.103e-01
	9.880e-01	1.400e+01	1.205e-01	1.949e+01	3.403e+03	1.320e+00	2.272e-02	3.404e+02	6.633e-13	0.000e+00
	1.014e-13	5.000e-01	5.480e-03	1.688e+01	3.198e+03	3.999e-01	2.533e-14	6.678e+00	0.000e+00	0.000e+00
	1.014e-13	0.000e+00	5.480e-03	1.688e+01	3.198e+03	3.999e-01	2.533e-14	5.145e+00	0.000e+00	0.000e+00
1000	1.247e+01	5.600e+01	9.684e+01	9.923e+01	9.729e+03	1.334e+01	3.943e+03	3.745e+07	3.486e-03	1.488e+01
	8.838e-01	1.450e+01	1.444e-01	1.832e+01	3.370e+03	1.400e+00	2.790e-02	1.238e+02	0.000e+00	0.000e+00
	1.014e-13	5.000e-01	5.476e-03	1.508e+01	3.139e+03	3.999e-01	2.533e-14	6.061e+00	0.000e+00	0.000e+00
	1.014e-13	0.000e+00	5.476e-03	1.508e+01	3.139e+03	3.999e-01	2.533e-14	4.895e+00	0.000e+00	0.000e+00
10 000	1.409e+01	6.350e+01	2.057e+02	1.187e+02	1.022e+04	1.950e+01	8.695e+03	8.583e+07	1.074e-02	9.538e+01
	3.119e+00	2.350e+01	1.146e+00	3.523e+01	4.811e+03	3.372e+00	2.473e+01	5.824e+03	3.223e-10	0.000e+00
	1.014e-13	5.000e-01	5.476e-03	1.417e+01	2.903e+03	3.999e-01	2.533e-14	5.973e+00	0.000e+00	0.000e+00
	1.014e-13	0.000e+00	5.476e-03	1.417e+01	2.903e+03	3.999e-01	2.533e-14	4.754e+00	0.000e+00	0.000e+00
100 000	1.421e+01	6.450e+01	2.285e+02	1.206e+02	1.023e+04	2.038e+01	8.898e+03	1.041e+08	9.427e-03	9.819e+01
	1.270e+01	5.100e+01	9.727e+01	9.411e+01	9.722e+03	1.498e+01	4.280e+03	7.347e+06	3.241e-03	7.581e-01
	3.242e-08	3.500e+00	5.476e-03	9.347e+00	1.757e+03	4.999e-01	2.533e-14	1.964e+01	0.000e+00	0.000e+00
	8.143e-14	0.000e+00	5.476e-03	8.744e+00	1.757e+03	3.999e-01	1.267e-14	4.823e+00	0.000e+00	0.000e+00
1 000 000	1.423e+01	6.450e+01	2.379e+02	1.209e+02	1.022e+04	2.065e+01	9.485e+03	1.088e+08	9.707e-03	9.662e+01
	1.313e+01	5.400e+01	1.198e+02	9.841e+01	9.781e+03	1.608e+01	5.161e+03	1.053e+07	3.256e-03	4.823e+00
	1.117e+01	3.700e+01	5.415e+01	7.784e+01	8.483e+03	1.165e+01	2.405e+03	1.990e+06	3.125e-04	7.470e-04
	6.071e-14	0.000e+00	8.207e-03	4.523e+00	1.303e+03	2.999e-01	1.267e-14	4.689e+00	0.000e+00	0.000e+00

Table C.2: [Median solution qualities achieved by the decreasing inertia weight optimizers with different schedules based on median solution qualities and a ring topology. Each cell contains the median solution quality achieved by the corresponding optimizer after 1 000, 10 000, 100 000 and 1 000 000 function evaluations.

Schedule	High dimensional								Low dimensional	
	Multimodal						Unimodal		Multimodal	Unimodal
	Ackley	Step	Griewank	Rastrigin	Schwefel	Salomon	Sphere	Rosenbrock	Schaffer	Easom
100	1.340e+01	5.700e+01	1.779e+02	1.027e+02	9.060e+03	1.416e+01	6.231e+03	1.289e+09	3.449e-03	8.657e+00
	3.660e+00	2.300e+01	1.297e+00	3.375e+01	4.869e+03	4.015e+00	2.989e+01	4.207e+04	3.239e-03	0.000e+00
	3.484e-09	4.500e+00	1.556e-14	1.930e+01	3.790e+03	4.999e-01	1.267e-14	8.752e+00	0.000e+00	0.000e+00
	2.000e-14	0.000e+00	0.000e+00	1.749e+01	3.632e+03	2.999e-01	0.000e+00	3.777e-01	0.000e+00	0.000e+00
1000	1.400e+01	6.050e+01	2.226e+02	1.131e+02	9.659e+03	1.733e+01	8.375e+03	1.080e+09	4.250e-03	4.465e+01
	3.798e+00	2.350e+01	1.314e+00	3.354e+01	5.097e+03	4.100e+00	3.123e+01	3.157e+04	3.239e-03	0.000e+00
	2.917e-09	4.500e+00	1.556e-14	1.779e+01	3.869e+03	4.999e-01	1.267e-14	1.745e+01	0.000e+00	0.000e+00
	2.000e-14	0.000e+00	0.000e+00	1.628e+01	3.790e+03	2.999e-01	0.000e+00	1.260e-01	0.000e+00	0.000e+00
10 000	1.461e+01	6.450e+01	3.585e+02	1.347e+02	1.021e+04	2.212e+01	1.365e+04	1.327e+09	1.286e-02	9.985e+01
	6.572e+00	3.150e+01	6.802e+00	4.561e+01	6.026e+03	6.094e+00	2.652e+02	1.690e+05	8.858e-04	0.000e+00
	8.854e-09	5.000e+00	1.556e-14	1.628e+01	3.830e+03	4.999e-01	1.267e-14	7.139e+00	0.000e+00	0.000e+00
	2.000e-14	0.000e+00	0.000e+00	1.568e+01	3.672e+03	2.999e-01	0.000e+00	1.022e-01	0.000e+00	0.000e+00
100 000	1.468e+01	6.500e+01	3.756e+02	1.380e+02	1.026e+04	2.293e+01	1.465e+04	1.308e+09	1.282e-02	1.000e+02
	1.302e+01	5.400e+01	1.087e+02	9.553e+01	9.322e+03	1.555e+01	4.804e+03	1.022e+07	3.239e-03	5.678e-01
	5.957e-03	8.500e+00	2.822e-04	1.417e+01	2.761e+03	1.100e+00	7.226e-06	2.741e+01	0.000e+00	0.000e+00
	2.000e-14	0.000e+00	0.000e+00	1.176e+01	2.685e+03	2.999e-01	0.000e+00	6.374e-02	0.000e+00	0.000e+00
1 000 000	1.469e+01	6.500e+01	3.766e+02	1.391e+02	1.023e+04	2.262e+01	1.446e+04	1.336e+09	1.248e-02	1.000e+02
	1.367e+01	5.750e+01	1.394e+02	1.022e+02	9.719e+03	1.706e+01	6.158e+03	1.650e+07	3.243e-03	2.671e+00
	1.028e+01	3.300e+01	3.715e+01	6.753e+01	7.407e+03	9.708e+00	1.547e+03	1.033e+06	1.970e-04	3.286e-04
	2.000e-14	0.000e+00	1.556e-14	8.744e+00	1.559e+03	2.999e-01	0.000e+00	1.126e+00	0.000e+00	0.000e+00

Table C.3: [Median solution qualities achieved by the decreasing inertia weight optimizers with different schedules based on median solution qualities and a square topology. Each cell contains the median solution quality achieved by the corresponding optimizer after 1 000, 10 000, 100 000 and 1 000 000 function evaluations.

Schedule	High dimensional								Low dimensional	
	Multimodal						Unimodal		Multimodal	Unimodal
	Ackley	Step	Griewank	Rastrigin	Schwefel	Salomon	Sphere	Rosenbrock	Schaffer	Easom
100	1.227e+01	5.350e+01	1.031e+02	9.127e+01	9.076e+03	1.197e+01	3.793e+03	2.924e+08	3.271e-03	2.353e+00
	2.054e+00	1.900e+01	6.438e-01	2.293e+01	4.396e+03	2.700e+00	3.691e+00	2.797e+03	1.197e-05	0.000e+00
	6.071e-14	2.500e+00	1.556e-14	1.477e+01	3.277e+03	2.999e-01	1.267e-14	1.957e+01	0.000e+00	0.000e+00
	2.000e-14	0.000e+00	1.556e-14	1.417e+01	3.158e+03	1.999e-01	1.267e-14	1.050e+00	0.000e+00	0.000e+00
1000	1.333e+01	6.000e+01	1.517e+02	1.032e+02	9.671e+03	1.534e+01	5.786e+03	2.372e+08	3.820e-03	4.518e+01
	2.216e+00	1.950e+01	6.546e-01	2.340e+01	4.306e+03	2.900e+00	4.147e+00	2.581e+03	1.479e-03	0.000e+00
	6.071e-14	2.500e+00	1.556e-14	1.327e+01	3.198e+03	2.999e-01	1.267e-14	1.852e+01	0.000e+00	0.000e+00
	2.000e-14	0.000e+00	1.556e-14	1.236e+01	3.119e+03	1.999e-01	1.267e-14	8.055e-01	0.000e+00	0.000e+00
10 000	1.442e+01	6.500e+01	2.830e+02	1.234e+02	1.021e+04	2.114e+01	1.096e+04	3.798e+08	1.245e-02	9.960e+01
	5.291e+00	2.800e+01	3.670e+00	4.167e+01	5.810e+03	5.011e+00	1.494e+02	5.204e+04	1.194e-03	0.000e+00
	1.421e-13	2.500e+00	4.109e-03	1.327e+01	3.237e+03	2.999e-01	1.267e-14	1.935e+01	0.000e+00	0.000e+00
	2.000e-14	0.000e+00	4.109e-03	1.206e+01	3.139e+03	1.999e-01	1.267e-14	7.849e-01	0.000e+00	0.000e+00
100 000	1.456e+01	6.500e+01	2.895e+02	1.285e+02	1.021e+04	2.165e+01	1.178e+04	4.294e+08	1.274e-02	9.989e+01
	1.289e+01	5.350e+01	1.002e+02	9.200e+01	9.478e+03	1.562e+01	4.533e+03	8.351e+06	3.240e-03	9.012e-01
	9.574e-05	7.000e+00	4.109e-03	1.086e+01	2.191e+03	6.999e-01	1.354e-08	2.169e+01	0.000e+00	0.000e+00
	2.000e-14	0.000e+00	1.556e-14	9.648e+00	2.152e+03	1.999e-01	1.267e-14	1.022e+00	0.000e+00	0.000e+00
1 000 000	1.454e+01	6.500e+01	2.933e+02	1.279e+02	1.021e+04	2.170e+01	1.198e+04	3.932e+08	1.273e-02	9.925e+01
	1.340e+01	5.600e+01	1.314e+02	9.969e+01	9.644e+03	1.680e+01	5.615e+03	1.262e+07	3.247e-03	3.899e+00
	1.048e+01	3.350e+01	3.870e+01	7.056e+01	7.581e+03	1.020e+01	1.688e+03	1.104e+06	2.524e-04	2.166e-04
	2.000e-14	0.000e+00	1.556e-14	6.332e+00	1.303e+03	2.999e-01	1.267e-14	2.957e+00	0.000e+00	0.000e+00

Bibliography

[1] D. H. Ackley. *A connectionist machine for genetic hillclimbing.* Kluwer, Boston, 1987.

[2] P. J. Angeline. Using selection to improve particle swarm optimization. In *Proceedings of the 1998 IEEE Congress on Evolutionary Computation*, pages 84–89, Piscataway, NJ, 1998. IEEE Press.

[3] T. Bäck. *Evolutionary algorithms in theory and practice.* Oxford University Press, 1996.

[4] T. Bartz-Beielstein. *Experimental Research in Evolutionary Computation. The New Experimentalism.* Springer, Berlin, Germany, 2006.

[5] E. Bonabeau, M. Dorigo, and G. Theraulaz. *Swarm Intelligence: From Natural to Artificial Systems.* Santa Fe Institute Studies on the Sciences of Complexity. Oxford University Press, USA, 1999.

[6] A. Chatterjee and P. Siarry. Nonlinear inertia weight variation for dynamic adaptation in particle swarm optimization. *Computers & Operations Research*, 33(3):859–871, 2006.

[7] M. Clerc and J. Kennedy. The particle swarm–explosion, stability, and convergence in a multidimensional complex space. *IEEE Transactions on Evolutionary Computation*, 6(1):58–73, 2002.

[8] P. R. Cohen. *Empirical Methods for Artificial Intelligence.* The MIT Press, 1995.

[9] M. Dorigo and T. Stützle. *Ant Colony Optimization.* Bradford Books. The MIT Press, 2004.

[10] E. E. Easom. A survey of global optimization techniques. Master's thesis, University of Louisville, Louisville, KY, 1990.

[11] R. Eberhart and J. Kennedy. A new optimizer using particle swarm theory. In *Proceedings of the 6th International Symposium on Micro Machine and Human Science*, pages 39–43, Piscataway, NJ, 1995. IEEE Press.

[12] R. Eberhart and Y. Shi. Comparing inertia weights and constriction factors in particle swarm optimization. In *Proceedings of the 2000 IEEE Congress on Evolutionary Computation*, pages 84–88, Piscataway, NJ, 2000. IEEE Press.

[13] R. Eberhart and Y. Shi. Tracking and optimizing dynamic systems with particle swarms. In *Proceedings of the 2001 IEEE Congress on Evolutionary Computation*, pages 94–100, Piscataway, NJ, 2001. IEEE Press.

[14] F. Ghannadian, C. Alford, and R. Shonkwiler. Application of random restarts to genetic algorithms. *Information Sciences*, 95:81–102, 1996.

[15] N. Higashi and H. Iba. Particle swarm optimization with gaussian mutation. In *Proceedings of the IEEE Swarm Intelligence Symposium 2003 (SIS'03)*, pages 72–79, Piscataway, NJ, 2003. IEEE Press.

[16] N. Holden and A. A. Freitas. A hybrid particle swarm/ant colony algorithm for the classification of hierarchical biological data. In *Proceedings of the IEEE Swarm Intelligence Symposium (SIS-2005)*, pages 100–107, Piscataway, NJ, 2005. IEEE Press.

[17] H. H. Hoos and T. Stützle. *Stochastic Local Search: Foundations and Applications*. Morgan Kaufmann Publishers, San Francisco, CA, USA, 2004.

[18] T. Jansen. On the analysis of dynamic restart strategies for evolutionary algorithms. In *Proceedings of the 7th International Conference on Parallel Problem Solving from Nature*, pages 33–43, Berlin, 2002. Springer-Verlag.

[19] S. Janson and M. Middendorf. A hierarchical particle swarm optimizer and its adaptive variant. *IEEE Transactions on Systems, Man and Cybernetics–Part B*, 35(6):1272–1282, 2005.

[20] C.-F. Juang. A hybrid of genetic algorithm and particle swarm optimization for recurrent retwork design. *IEEE Transactions on Systems, Man and Cybernetics, Part B*, 34(2):997–1006, 2004.

[21] J. Kennedy. The particle swarm: Social adaptation of knowledge. In *Proceedings of the 1997 International Conference on Evolutionary Computation*, pages 303–308, 1997.

[22] J. Kennedy. Methods of agreement: Inference among the eleMentals. In *Proceedings of the 1998 IEEE International Symposium on Intelligent Control*, pages 883–887, 1998.

[23] J. Kennedy and R. Eberhart. Particle swarm optimization. In *Proceedings of IEEE International Conference on Neural Networks*, pages 1942–1948, Piscataway, NJ, 1995. IEEE Press.

[24] J. Kennedy, R. Eberhart, and Y. Shi. *Swarm Intelligence*. Morgan Kaufmann, 2001.

[25] J. Kennedy and R. Mendes. Population structure and particle swarm performance. In *Proceedings of the 2002 IEEE Congress on Evolutionary Computation*, pages 1671–1676, Piscataway, NJ, 2002. IEEE Press.

[26] S. Kern, S. D. Müller, N. Hansen, D. Büche, J. Ocenasek, and P. Koumoutsakos. Learning probability distributions in continuous evolutionary algorithms–a comparative review. *Natural Computing*, 3(1):77–112, 2004.

[27] M. Løvbjerg, T. K. Rasmusen, and T. Krink. Hybrid particle swarm optimiser with breeding and subpopulations. In *Proceedings of the third Genetic and Evolutionary Computation Conference (GECCO-2001)*, pages 469–476, San Francisco, CA, USA, 2001. Morgan Kaufmann.

[28] M. Luby, A. Sinclair, and D. Zuckerman. Optimal speedup of las vegas algorithms. In *Proceedings of the 2nd Israel Symposium on Theory of Computing and Systems*, pages 128–133, 1993.

[29] S. Luke. When short runs beat long runs. In L. Spector, E. D. Goodman, A. Wu, W. B. Langdon, H.-M. Voigt, M. Gen, S. Sen, M. Dorigo, S. Pezeshk, M. H. Garzon, and E. Burke, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, pages 74–80, San Francisco, California, USA, 7-11 2001. Morgan Kaufmann.

[30] M. Meissner, M. Schmuker, and G. Schneider. Optimized particle swarm optimization (opso) and its application to artificial neural network training. *BMC Bioinformatics*, 7(125), 2006.

[31] R. Mendes. *Population Topologies and Their Influence in Particle Swarm Performance.* PhD thesis, Escola de Engenharia, Universidade do Minho, 2004.

[32] R. Mendes, J. Kennedy, and J. Neves. Watch thy neighbor or how the swarm can learn from its environment. In *Proceedings of the IEEE Swarm Intelligence Symposium (SIS-2003)*, pages 88–94, Piscataway, NJ, 2003. IEEE Press.

[33] R. Mendes, J. Kennedy, and J. Neves. The fully informed particle swarm: Simpler, maybe better. *IEEE Transactions on Evolutionary Computation*, 8(3):204–210, 2004.

[34] A. Mohais, R. Mendes, C. Ward, and C. Postoff. Neighborhood restructuring in particle swarm optimization. In S. Zhang and R. Jarvis, editors, *LNCS 3809. Proceedings of the 18th Australian Joint Conference on Artificial Intelligence*, pages 776–785, Berlin, 2005. Springer.

[35] D. Ortiz-Boyer, C. Herbás-Martínez, and N. Garciá-Pedrajas. CIXL2 - A crossover operator for evolutionary algorithms based on population features. *Journal of Artifical Intelligence Research*, 24:1–48, 2005.

[36] E. Ozcan and C. K. Mohan. Analysis of a simple particle swarm optimization system. *Intelligent Engineering Systems Through Artificial Neural Networks*, 8:253–258, 1998.

[37] E. Ozcan and C. K. Mohan. Particle swarm optimization: Surfing the waves. In *Proceedings of the 1999 IEEE Congress on Evolutionary Computation*, Piscataway, NJ, 1999. IEEE Press.

[38] M. Pelikan, D. E. Goldberg, and F. Lobo. A survey of optimization by building and using probabilistic models. *Computational Optimization and Applications*, 21(1):5–20, 2002.

[39] R. Poli, C. D. Chio, and W. B. Langdon. Exploring extended particle swarms: A genetic programming approach. In *Proceedings of the 2005 conference on Genetic and evolutionary computation*, pages 169–176, New York, NY, USA, 2005. ACM Press.

[40] R. Poli, W. B. Langdon, and O. Holland. Extending particle swarm optimisation via genetic programming. In M. Keijzer, A. Tettamanzi, P. Collet, J. van Hemert, and M. Tomassini, editors, *LNCS 3447. Proceedings of the 8th European Conference on Genetic Programming: EuroGP 2005*, pages 291–300, Berlin, 2005. Springer.

[41] L. A. Rastrigin. *Systems of extremal control.* Nauka, Moscow, 1974.

[42] 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(3):240–255, 2004.

[43] W. T. Reeves. Particle systems–a technique for modeling a class of fuzzy objects. *ACM Transactions on Graphics*, 2(2):91–108, 1983.

[44] C. W. Reynolds. Flocks, herds, and schools: A distributed behavioral model. *ACM Computer Graphics*, 21(4):25–34, 1987.

[45] H. H. Rosenbrock. An automatic method for finding the greatest or least value of a function. *The Computer Journal*, 3(3):175–184, 1960.

[46] R. Salomon. Reevaluating genetic algorithm performance under coordinate rotation of benchmark functions. A survey of some theoretical and practical aspects of genetic algorithms. *BioSystems*, 39(3):263–278, 1996.

[47] B. R. Secrest and G. B. Lamont. Visualizing particle swarm optimization - gaussian particle swarm optimization. In *Proceedings of the IEEE Swarm Intelligence Symposium 2003 (SIS-2003)*, pages 198–204, Piscataway, NJ, 2003. IEEE Press.

[48] Y. Shi and R. Eberhart. A modified particle swarm optimizer. In *Proceedings of the IEEE International Conference on Evolutionary Computation*, pages 69–73, Piscataway, NJ, 1998. IEEE Press.

[49] Y. Shi and R. Eberhart. Parameter selection in particle swarm optimization. In *LNCS 1447. Proceedings of the 7th International Conference on Evolutionary Programming VII*, pages 591–600, Berlin, 1998. Springer-Verlag.

[50] Y. Shi and R. Eberhart. Empirical study of particle swarm optimization. In *Proceedings of the 1999 IEEE Congress on Evolutionary Computation*, pages 1945–1950, Piscataway, NJ, 1999. IEEE Press.

[51] K. Socha. ACO for Continuous and Mixed-Variable Optimization. In M. Dorigo, M. Birattari, and C. Blum, editors, *Proceedings of ANTS 2004 – Fourth International Workshop on Ant Colony Optimization and Swarm Intelligence*, volume 3172 of *LNCS*, pages 25–36. Springer-Verlag, Berlin, Germany, 5-8 September 2004.

[52] K. Socha and M. Dorigo. Ant colony optimization for continuous domains. Technical Report TR/IRIDIA/2005-037, IRIDIA, Université Libre de Bruxelles, December 2005.

[53] A. Stacey, M. Jancic, and I. Grundy. Particle swarm optimization with mutation. In *Proceedings of the 2003 IEEE Congress on Evolutionary Computation*, pages 1425–1430, Piscataway, NJ, 2003. IEEE Press.

[54] P. N. Suganthan. Particle swarm optimiser with neighbourhood operator. In *Proceedings of the 1999 IEEE Congress on Evolutionary Computation*, pages 1958–1962, Piscataway, NJ, 1999. IEEE Press.

[55] P. N. Suganthan, N. Hansen, J. J. Liang, K. Deb, Y.-P. Chen, A. Auger, and S. Tiwari. Problem definitions and evaluation criteria for the CEC 2005 special session on real-parameter optimization. Technical Report 2005005, Nanyang Technological University, Singapore and IIT Kanpur, India, May 2005.

[56] I. C. Trelea. The particle swarm optimization algorithm: Convergence analysis and parameter selection. *Information Processing Letters*, 85(6):317–325, 2003.

[57] D. Whitley, S. B. Rana, J. Dzubera, and K. E. Mathias. Evaluating evolutionary algorithms. *Artificial Intelligence*, 85(1-2):245–276, 1996.

[58] Y.-L. Zheng, L.-H. Ma, L.-Y. Zhang, and J.-X. Qian. Empirical study of particle swarm optimizer with an increasing inertia weight. In *Proceedings of the 2003 IEEE Congress on Evolutionary Computation*, pages 221–226, Piscataway, NJ, 2003. IEEE Press.

[59] Y.-L. Zheng, L.-H. Ma, L.-Y. Zhang, and J.-X. Qian. On the convergence analysis and parameter selection in particle swarm optimization. In *Proceedings of the 2003 IEEE International Conference on Machine Learning and Cybernetics*, pages 1802–1807, Piscataway, NJ, 2003. IEEE Press.