Cooperative Models of Particle Swarm Optimizers

by

Mohammed El-Abd

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2008

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Particle Swarm Optimization (PSO) is one of the most effective optimization tools, which emerged in the last decade. Although, the original aim was to simulate the behavior of a group of birds or a school of fish looking for food, it was quickly realized that it could be applied in optimization problems. Different directions have been taken to analyze the PSO behavior as well as improving its performance. One approach is the introduction of the concept of cooperation. This thesis focuses on studying this concept in PSO by investigating the different design decisions that influence the cooperative PSO models' performance and introducing new approaches for information exchange.

Firstly, a comprehensive survey of all the cooperative PSO models proposed in the literature is compiled and a definition of what is meant by a cooperative PSO model is introduced. A taxonomy for classifying the different surveyed cooperative PSO models is given. This taxonomy classifies the cooperative models based on two different aspects: the approach the model uses for decomposing the problem search space and the method used for placing the particles into the different cooperating swarms. The taxonomy helps in gathering all the proposed models under one roof and understanding the similarities and differences between these models.

Secondly, a number of parameters that control the performance of cooperative PSO models are identified. These parameters give answers to the four questions: Which information to share? When to share it? Whom to share it with? and What to do with it? A complete empirical study is conducted on one of the cooperative PSO models in order to understand how the performance changes under the influence of these parameters.

Thirdly, a new heterogeneous cooperative PSO model is proposed, which is based on the exchange of probability models rather than the classical migration of particles. The model uses two swarms that combine the ideas of PSO and Estimation of Distribution Algorithms (EDAs) and is considered heterogeneous since the cooperating swarms use different approaches to sample the search space. The model is tested using different PSO models to ensure that the performance is robust against changing the underlying population topology. The experiments show that the model is able to produce better results than its components in many cases. The model also proves to be highly competitive when compared to a number of state-of-the-art cooperative PSO algorithms.

Finally, two different versions of the PSO algorithm are applied in the FPGA placement problem. One version is applied entirely in the discrete domain, which is the first attempt to solve this problem in this domain using a discrete PSO (DPSO). Another version is implemented in the continuous domain. The PSO algorithms are applied to several well-known FPGA benchmark

problems with increasing dimensionality. The results are compared to those obtained by the academic Versatile Place and Route (VPR) placement tool, which is based on Simulated Annealing (SA). The results show that these methods are competitive for small and medium-sized problems. For higher-sized problems, the methods provide very close results. The work also proposes the use of different cooperative PSO approaches using the two versions and their performances are compared to the single swarm performance.

Acknowledgements

This thesis would not be possible without the support of many individuals, to whom I would like to express my gratitude. I will always be indebted to my supervisor, Prof. Mohamed Kamel, for his key role in my development as a person and as a researcher. He offered me support, encouragement, guidance, and most importantly trust. His input and guidance was invaluable to the quality and contribution of the work presented in this thesis, as well as in other publications. His trust and support was instrumental in giving me confidence to achieve many accomplishments.

I wish to thank many of my colleagues at the Pattern Analysis and Machine Intelligence (PAMI) Lab, especially Shady Shehata, Abbas Ahmadi and Moataz El Ayadi, for valuable discussions and feedback.

I also wish to thank Hassan Hassan for being both a great friend and a valuable collaborative worker during my studies.

I would like to thank the PAMI administrative secretary Heidi Campbell for her help and support during my graduate studies.

I would like to thank Ahmed Abdel Rahman, Mohamed El-Said, Ismael El-Samahy, Khaled Hammouda, Mohamed Hassan, Ayman Ismail, Hazem Shehata, Shady Shehata, Nayer Wanas, Ahmed Youssef, and Hatem Zeineldin, for being such great friends.

Finally I would like to thank my wife, Walaa, for her unconditional support and love, without which many things would not be possible. I would like also to thank my mother Foaz, my late father Hamed, my sister Safia, and my brother Mostafa, for their support and encouragement throughout my life.

Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

1.1 Motivation

Although a number of different Cooperative Particle Swarm Optimizers (CPSO) have emerged in the past 6 years in order to efficiently solve larger problems. The work in this field still falls short in different directions:

- Many heuristic search methods have been used in a cooperative search environment before PSO including Tabu Search (TS), Genetic Algorithms (GAs), and Ant Colony Optimization (ACO). In these implementations, there are a number of parameters that need to be tuned in order to have a powerful cooperative system. Some of these parameters have been studied before in different areas. However, such studies have never been fully carried out for PSO cooperative models.

- Parallel Estimation of Distribution Algorithms (EDAs) is a new research direction that has been pursued in the previous 4 years. In this field, it was shown that exchanging information on the form of probabilistic models capturing the search space characteristics can produce better results than the classical migration of individuals. However, almost all cooperative PSO models proposed up to-date rely on exchanging information in the form of particles. This motivated the investigation of using such a powerful exchange scheme in cooperative PSO.

- To follow on the previous point, the parallel EDAs proposed so far had all the populations using the same probability models to sample the search space. Hence, the different populations might actually run into the same problems caused by the used model. Nevertheless, these

approaches managed to produce better results for a different number of cases. Employing a heterogeneous cooperative model where the cooperating swarms use different approaches in sampling the search space is an interesting direction to follow as one could end-up using the benefits of the different approaches.

- To our knowledge, there haven't been any cooperative implementations of a discrete version of PSO. This direction is investigated in this thesis by taking the FPGA placement problem as the application under study.

1.2 Proposed Work

The contributions of this work is summarized as follows:

- The work gives a comprehensive survey of all the cooperative PSO models proposed in the literature. These models are categorized by the application they were designed for.
- The work proposes a taxonomy for classifying the different surveyed cooperative PSO models. This taxonomy classifies the cooperative models based on two different aspects: the decomposition approach adopted by the model and the method used for placing the particles into the different cooperating swarms.
- The work gives a definition of what is meant by a cooperative model. This definition helps in identifying key design issues that are essential in having a successful model. These decisions give answers to the four questions: Which information to share? When to share it? Whom to share it with? and What to do with it? The design decisions taken by all the surveyed cooperative PSO models are identified which helps to shed light on the similarities and differences between these models.
- The work performs a complete empirical study on one of the cooperative PSO models in order to understand how the performance changes under the influence of the design issues previously identified. The addressed issues include the exchange of the *gbest* (global best) information vs. the exchange of complete particles, changing the number of iterations between successive communication steps, changing the number of cooperating swarms, changing the method for selection and replacement of exchanged particles and changing the number of exchanged particles.

- The work proposes a new heterogeneous cooperative PSO model, which is based on the exchange of probability models rather than the classical migration of particles. The model uses two swarms that combine the ideas of PSO and EDAs. Since the two swarms use two different probability models, each swarm performs a *model conversion* step on the received probability model to transform it into a model similar to its own. After that, a *model combination* step is performed between the resident and received models before continuing with the search.

- The work investigates the idea of applying a Discrete PSO (DPSO) algorithm for the FPGA placement problem. This is the first attempt to entirely solve this problem in the discrete domain using PSO. The work also proposes the use of a discrete cooperative PSO (DCPSO) version by optimizing the placement of different types of blocks by different swarms.

All the experiments conducted in this thesis are implemented on an Intel Xeon machine with a 3.06GHz CPU and a 1.00GB of RAM. All the codes have been implemented using VC++ 6.0.

1.3 Thesis Organization

This work is organized as follows: an overview of the cooperative PSO models proposed in the literature presented in Chapter 2. In Chapter 3, the work defines what is meant by a cooperative PSO model, points out the key parameters associated withe their design, identifies the design decisions taken by previously proposed models and conducts a complete empirical study on one of theses models. A new PSO and EDA hybrid is introduced in Chapter 4. In Chapter 5, the heterogeneous PSO-EDA cooperative optimizer is proposed and studied. Chapter 6 investigates the application of both discrete and continuous PSO and their cooperative versions to the FPGA placement problem. Conclusions and future work are presented in Chapter 7.

Chapter 2

A Taxonomy of Cooperative Particle Swarm Optimizers

This chapter starts by giving a brief background about PSO. A comprehensive survey of all the cooperative PSO models proposed in the literature is then presented. A taxonomy for classifying the different surveyed cooperative PSO models is proposed. This taxonomy classifies the cooperative models based on two different aspects: the decomposition approach adopted by the model and the method used for placing the particles into the different cooperating swarms. The taxonomy helps in gathering all the proposed models under one roof and understanding the similarities and differences between these models.

2.1 Particle Swarm Optimization

The PSO [1,2] method is regarded as a population-based method, where the population is referred to as a swarm. The swarm consists of a number of individuals called particles. Each particle i in the swarm holds the following information: (i) the current position x_i, (ii) the current velocity v_i, (iii) the best position, the one associated with the best fitness value the particle has achieved so far $pbest_i$, and (iv) the global best position, the one associated with the best fitness value found among all of the particles $gbest$. In every iteration, each particle adjusts its own trajectory in the space in order to move towards its best position and the global best according to the following equations:

$$\begin{aligned} v_{ij}^{t+1} = w v_{ij}^{t} + c_1 r_{1j}^{t}(pbest_{ij}^{t} - x_{ij}^{t}) \\ + c_2 r_{2j}^{t}(gbest_{j}^{t} - x_{ij}^{t}), \end{aligned} \tag{2.1}$$

$$x_{ij}^{t+1} = x_{ij}^{t} + v_{ij}^{t+1}, \tag{2.2}$$

for $j \in 1..d$ where d is the number of dimensions, $i \in 1..n$ where n is the number of particles, t is the iteration number, w is the inertia weight, r_1 and r_2 are two random numbers uniformly distributed in the range [0,1], and c_1 and c_2 are the acceleration factors.

Afterwards, each particle updates its personal best using the equation (assuming a minimization problem):

$$pbest_i^{t+1} = \begin{cases} pbest_i^t & \texttt{if} \quad f(pbest_i^t) \leq f(x_i^{t+1}) \\ x_i^{t+1} & \texttt{if} \quad f(pbest_i^t) > f(x_i^{t+1}) \end{cases} \tag{2.3}$$

Finally, the global best of the swarm is updated using the equation (assuming a minimization problem):

$$gbest^{t+1} = \arg \min_{pbest_i^{t+1}} f(pbest_i^{t+1}), \tag{2.4}$$

where *f(.)* is a function that evaluates the fitness value for a given position. This model is referred to as the *gbest* (global best) model.

2.2 Cooperative PSO Models

Applying cooperation in PSO followed the same steps that were taken in applying cooperation in any other search algorithm. Cooperative search algorithms have been extensively studied in the past decade to effectively solve many large size optimization problems. The basic approach involves having more than one search module running and exchanging information among each other in order to explore the search space more efficiently and reach better solutions.

Many heuristic search methods have been used in a cooperative search environment including Tabu Search [3, 4], Genetic Algorithms [5, 6], Ant Colony Optimization [7, 8] and Particle Swarm Optimization [9, 10].

Several cooperative models have been introduced for PSO in the past few years. These models are surveyed in this section and categorized by the application they were used in.

2.2.1 Single-Objective Optimization

2.2.1.1 Static Optimization

A cooperative approach referred to as Cooperative PSO (CPSO_S), was introduced in [9, 11]. The approach relies on splitting the space (solution vector) into sub-spaces (smaller vectors) where

each sub-space is optimized using a separate swarm. The overall solution vector is constructed using the solutions found by the best particle of each swarm. To update the fitness value for a certain particle i in a swarm j, a solution vector is used with that particle and the best particles of all the other swarms. This approach was originally introduced using genetic algorithms [12]. This approach is illustrated in Figure 2.1.

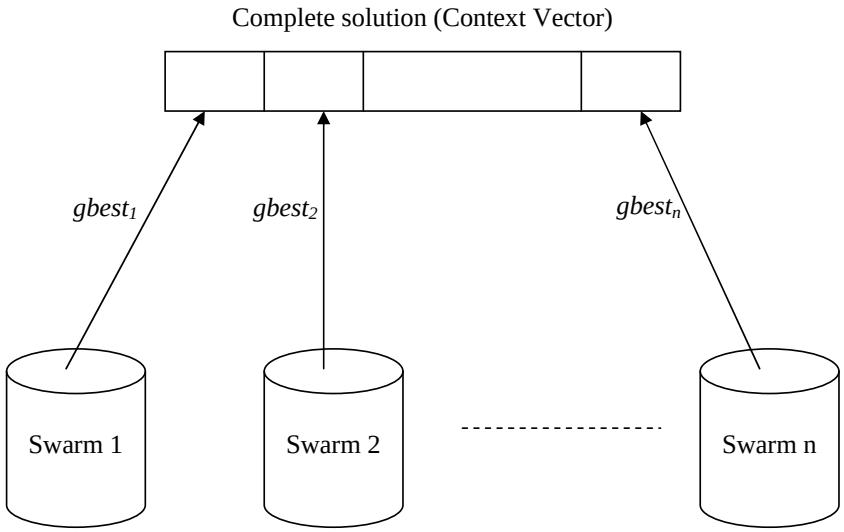


Figure 2.1: The CPSO_S approach.

A hybrid cooperative approach was also introduced in [9, 11], it was referred to as the hybrid CPSO (CPSO_H). It consists of having two search stages working in a serial fashion. Each stage was only run for one iteration then passing the best found solution to the next stage. The first stage applied the CPSO_S technique and the second stage used the normal PSO algorithm. This approach was applied to take advantage of the ability of PSO to escape pseudo-minimizers while benefiting from the CPSO_S fast convergence property. This approach is shown in Figure 2.2.

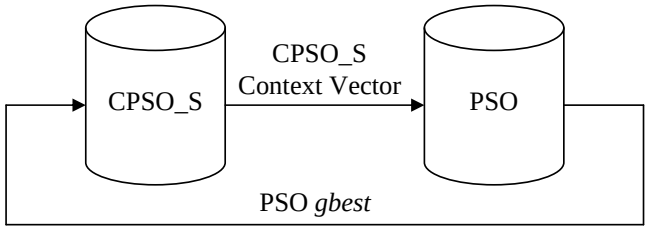


Figure 2.2: The CPSO_H approach.

A different cooperative approach was introduced in [10], referred to as concurrent PSO (CONPSO). The approach adopted was to have two swarms searching concurrently for a solution with frequent

message passing of information. The information exchanged was the global bests of the two swarms. After every exchange point, the two swarms were to track the better global best found. The two swarms were using two different approaches, one adopted the original PSO method, and the other used the Fitness-to-Distance Ratio PSO (FDRPSO) [13]. This approach improved the performance over both methods as well as minimizing the time requirement of the FDRPSO alone. Figure 2.3 illustrates this approach.

Figure 2.3: The CONPSO approach.

In [14, 15], a Parallel PSO (PPSO) was proposed. The idea was similar to [10] but with using more than two swarms, referred to as *groups*, and all performing the same PSO algorithm. The exchange of information was also performed every predetermined number of iterations. The authors proposed changing the method the information is exchanged depending on the level of correlation between the problem variables. If the variables are uncorrelated or loosely correlated, the overall *gbest* value is shared among all the *groups*, mutated and then used to replace the poor particles. If the problem variables are strongly correlated, the exchange is done in a directed ring fashion. Finally, if the correlation information is unknown, a hybrid method is adopted.

In [16], the authors introduced a hierarchal cooperative particle swarm optimizer. This cooperative model was based on combining the CONPSO and CPSO_S models. The combination is achieved by having two swarms searching for a solution concurrently, each swarm is adopting the CPSO_S technique. Experiments run on four benchmark optimization functions showed that this approach produces better results than the CPSO_S ,CPSO_H and the CPSO_S model with multiple restarts in some cases. Results also showed that choosing a suitable synchronization period is related to the convergence behavior of CPSO_S.

In [17], the cooperating swarms exchanged information based on a diversity strategy. At the beginning of the search, the population was clustered into m different sub-swarms. At each communication stage, each swarm prepares a list of particles to be sent to another swarm and a list of particles to be replaced. The communication only occurs between swarms in the same neighborhood, determined by an inter-swarm distance measure. Since distance information changes between communication stages, a sub-swarm could communicate with different sub-swarms every

time. The velocity update equation was also modified allowing each particle to follow the best particle in its sub-swarm, the best particle in its neighborhood and the best particle in the whole population. The authors concluded that the performance of their algorithm depends on the rate of information exchange and stated that more experiments need to be conducted to find an optimal exchange rate.

Different parallel PSO models were studied in [18]. The authors experimented with three different parallel versions of PSO. First, the master/slave PSO, which is a simple parallelization approach where the PSO operation is handled by a master processor that delegates some of the work to several slave processors. Second, the migration PSO, in which different swarms are run on different processors, and after a finite number of iterations the best solution of each swarm (processor) is migrated to the neighboring swarms (processors). Finally, the diffusion PSO, in which each particle is handled by a separate processor. Each particle has only local information about the best position achieved by its neighborhood. The neighborhood topology used was the Von-Neuman model [19]. Based on the complexity analysis, the authors came to the conclusion that the diffusion PSO can be regarded as a limiting case to the migration PSO. Also, the diffusion model is scalable compared to the master/slave model. Finally, the convergence rate of the diffusion model is dependent on the neighborhood topology used.

A multi-population cooperative PSO (MCPSO) was proposed in [20]. The authors adopted the master/slave approach by having one master swarm and several slave swarms. The slave swarms were to evolve in parallel then supply their best solutions to the master swarm. The master swarm then updates its particles by taking into account the information of the best solution among all the slave swarms. This information was integrated as a third component in the velocity update equation. This approach is illustrated in Figure 2.4, where *gbest* is the best solution among all the received ones.

In all of the previous implementations, all the swarms were *static*. If a particle is assigned to a specific swarm in the beginning of the search, it stays in that swarm till the end. A dynamic multi-swarm approach was presented in [21, 22]. In this approach, each swarm adopted the *lbest* model. After a predefined number of iterations k, the *regrouping period*, each particle gets randomly assigned to a different swarm. The information exchange in this approach is implicit rather than explicit since every particle takes its information and carries it when it is assigned to a different swarm.

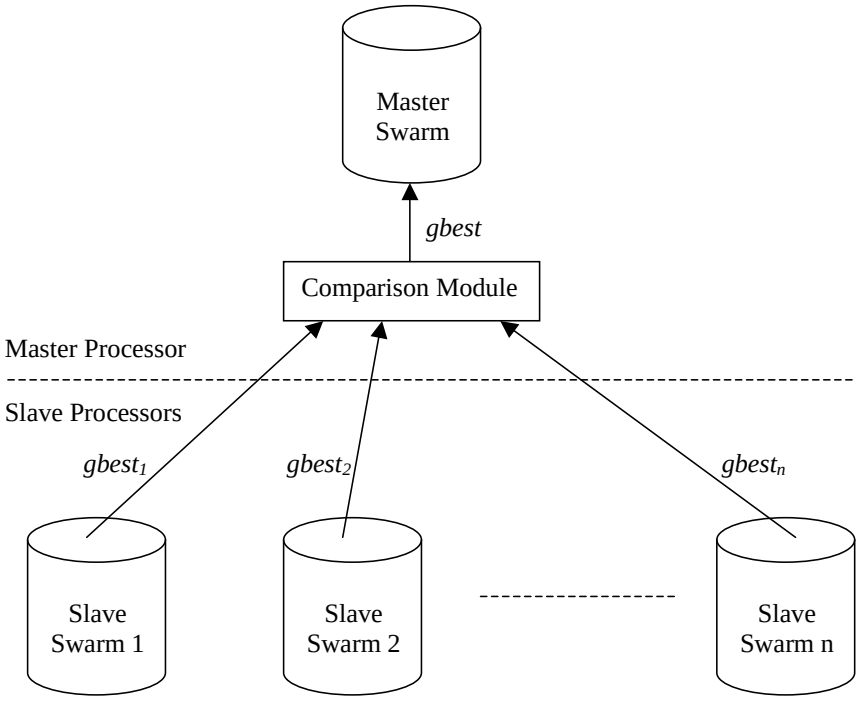


Figure 2.4: The MCPSO approach

2.2.1.2 Multimodal Optimization

To locate multiple optima in multimodal functions, a multi-swarm approach was presented in [23], referred to as the NichePSO. The approach was used to locate multiple optimal solutions in multimodal problems. Multiple sub-swarms were grown out of an initial main swarm. These sub-swarms utilized the guaranteed convergence PSO (GCPSO) algorithm [24]. These sub-swarms were allowed to merge if they intersect. They also had the ability to absorb new particles. New sub-swarms were created if a particle's fitness showed little change over a small number of iterations. The approach was applied to 5 different multimodal functions and it was able to locate all maxima in all the runs.

A speciation-based PSO (SPSO) was proposed in [25]. In SPSO, the population was dynamically divided into multiple species after identifying the *species seeds*. The seeds were identified after each step based on a similarity measure. Each seed was used as the neighborhood best in its species. SPSO was proven effective when dealing with multimodal functions with low dimensionality. The approach was further enhanced in [26] by using a time-based convergence measure in order to overcome the burden of specifying the species radius.

In [27], the authors propose a Multi-Grouped PSO (MGPSO) for multimodal function opti-

mization. To ensure that every group is approaching a different optimum. A repulsive velocity component is added to the particle's velocity equation if it becomes too close to the *gbest* of another group rather than its own. Also, the updating of the *gbest* values of the different groups is done in a way that ensure that these values are far enough from each other. The method was successfully applied to a number of problems showing a good performance when the number of groups is less or higher than the number of peaks.

2.2.1.3 Dynamic Optimization

In [28], a multi-swarms technique was tested on a number of dynamic multimodal benchmark functions. The swarms evolved in parallel and communicated with each other after every iteration to test if their attractors are within a certain range, the *exclusion radius*. If two swarms are following two attractors that are close to each other, the swarm that has the bad attractor gets all its particles positions and velocities re-initialized. In their experiments, the authors changed the number of swarms while fixing the total number of particles and the total number of performed function evaluations. They concluded that a multi-PSO model is better than a single swarm model and that the number of swarms is related to the number of optima. They also showed that the off-line error is reduced while increasing the number of swarms up to a certain limit after which the error starts to increase again.

In [29], the authors proposed the use of a partitioned hierarchical particle swarm optimizer (PH-PSO) to tackle dynamic optimization problems. Hierarchical PSO (HPSO) was first introduced in [30], where the particles were arranged in a dynamic hierarchical order that defines a dynamic neighborhood structure. Particles were allowed to move up and down that hierarchy so better particles can influence the swarm. In the cooperative approach, this hierarchy was divided into sub-swarms after a change occurs (defined as a change in the fitness value of the *gbest*). These sub-swarms search for the optimum in an independent fashion for a small number of iterations, the *division interval*, after which they get rejoined again. When this hierarchy is split again, the sub-swarms will not contain the same particles as before since these particles continuously move up and down the hierarchy. Experiments showed that PH-PSO performed best on multimodal functions where the changes were not too severe. The authors also proposed an adaptive version (PH-PSO-a), where the division interval is adaptively determined according to the optimization behavior. Different methods for determining the best time when to rejoin the sub-swarms and how to handle the topmost sub-swarm were studied in [31].

The authors in [32] developed a particle swarm model for tracking multiple maxima using

speciation in continuously dynamic environments. The model allowed parallel sub-populations to track different peaks. The best particle in each sub-population or species is referred to as the *species seed*. To ensure that the model is unbiased for local maxima, the species are reconstructed after every iteration, usually with different members and different seeds. The number of members per species was also limited to prevent many particles from exploiting the same peak. The experiments showed that low species populations has lower error rates in highly dynamic environments. It was also shown that in order to achieve the lowest average error, there is a maximum limit for the species size. This work was further extended in [33] by adding a mechanism to remove redundant particles.

A multi-swarms technique with multiple interaction strategies was proposed in [34]. The population was split into different sub-swarms. Nearby swarms interacted through an *exclusion* strategy similar to what was used in [28]. Another interaction also occurs globally among all the swarms through an *anti-convergence* operator. The diversity was maintained inside each swarm by using different types of particles. When changing the number of swarms, the authors reached the same conclusion as in [28]. The behavior of the multiswarms model was not so sensitive against changing the *exclusion radius*. In addition, a formula was provided for tuning this parameter. Experiments showed that the model performed well on a wide range of problems and was robust against changing the different parameters.

2.2.2 Multi-Objective Optimization

In [35], the authors introduced the parallel vector evaluated PSO (VEPSO) that was used to solve multi-objective optimization problems. Each swarm was optimizing a single objective function. The information of this function is exchanged with neighboring swarms via the exchange of the best experience of the swarm. The authors experimented with both the single-node (all the swarms on the same CPU) and the parallel (a single swarm per CPU) approaches using the ring topology. The parallel implementation provided better execution times. However, increasing the number of CPUs over six resulted in increased running times due to the communication overhead. The two swarms case was investigated separately in [36].

The autonomous agent response learning problem was addressed in [37] using a multi-species PSO (MS-PSO). The award function was divided into several award functions, hence, the response extraction process is modeled as a multi-objective optimization problem. Each objective function was solved using a different swarm. The information exchange process occurred between neighboring swarms and involved the best particles information. This was done after every iteration and

the velocity update equation was modified by taking into account the incoming information from all neighboring swarms.

Another Multi-objective PSO (AMOPSO) was introduced in [38]. In this approach, each sub-swarm performs a predetermined number of iterations, then the sub-swarms exchange information. This is done by grouping all the leaders in a single set. This set is again divided into groups, and each resulting group is assigned to a different swarm. The splitting is done with respect to the closeness in the decision variable space.

In [39], the authors proposed two different parallel versions of a Multi-Objective PSO (MOPSO). The basic idea was to have different sub-swarms running on different processors and after a few iterations these swarms report their best solutions to a central processor. The central processor then assigns each sub-swarm a *guide* and each sub-swarm re-initializes its particles in the local neighborhood of its assigned *guide* while including the *guide* in its particles. The authors experimented with two methods of assigning the guides: Cluster-based (CMOPOS) and Hypervolume-based (HMOPSO). The Hypervolume-based approach was found to produce better results.

2.2.3 Constrained Optimization

A co-evolutionary PSO for constrained optimization problems was proposed in [40]. The authors transformed the constrained problem into a min-max problem which was solved using a co-evolutionary approach. Two cooperating swarms were used, one swarm optimizing the min part of the problem and another swarm optimizing the max part. The two swarms exchanged information during the fitness evaluation process. In this model, one swarm is active at a time. The first swarm evolves for a predetermined number of iterations and uses the particles information of the second swarm during fitness evaluation. Then, this swarm is stopped and the second swarm evolves in the same manner using the particles information of the first swarm in the fitness evaluation process, and so on. However, the authors found it difficult to fine tune the solution using a uniform distribution. This problem was addressed in [41] by adopting the Gaussian probability distribution in generating the random numbers for updating the particles velocities. Figure 2.5 illustrates the idea behind this approach.

2.2.4 Other Applications

In [42], the CPSO was used to train product unit neural networks. The solution of the problem was the vector containing all the weights of the network. This vector was decomposed among several

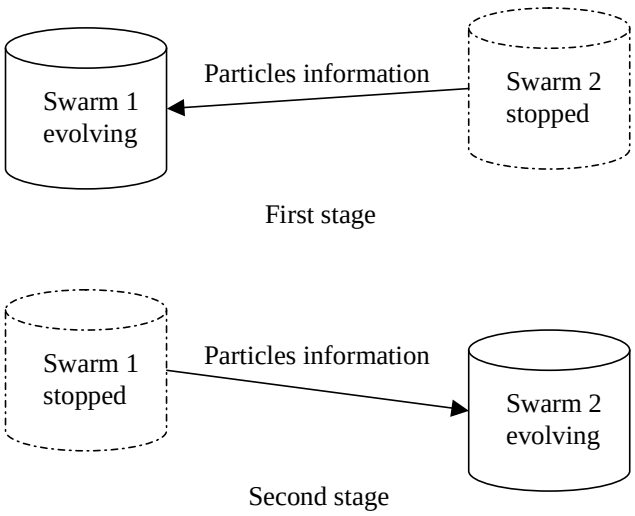


Figure 2.5: The co-evolving PSO approach

swarms, each swarm optimizing a specified number of weights. The authors studied the effect of changing the number of swarms, referred to as the *split factor*, on the training performance. They concluded that the training performance improves with increasing the split factor until a critical ratio is reached. This ratio was found to be $W/5$, where W is the total number of the optimized weights.

In [43], improvised music was played using a multi-swarm approach. Each swarm represented a musical entity and the particles in the swarm were musical events. The system as a whole was regarded as an improvising ensemble. Each particle was a 3-dimensional vector in the music space representing loudness, pulse and pitch. The ability of each individual to produce a coherent improvisation was ensured by the principles of self-organization.

In [44], the authors proposed a co-evolutionary PSO approach for solving the game of seega. Seega is an Egyptian two-stage board game. In the first stage, the two players take turns in placing their disks on the board until there is only one empty cell. In the second stage, the players take turns in moving their disks; if a disk, that belongs to one player, gets surrounded by disks of the other player, it gets captured and removed from the board. The game was solved by evolving two independent swarms representing the two players. The system consisted of two parts, the game engine and the co-evolutionary part. The second part used the game engine in a master-slave relationship in order to evaluate the particles fitness. The authors used the same approach proposed in [40] by using one swarm for each player.

The authors of [45] proposed a co-evolutionary PSO approach for tuning the parameters of a 5 degree-of-freedom arm robot torque controller. Two swarms evolved together, one swarm is optimizing the feedforward controller parameters and the other swarm is searching for the disturbance values in the worst case. The final solution is generated by both swarms. The two swarms were implemented in a serial fashion similar to the one used in [40]. Closed-loop simulations showed that the proposed strategy improved the trajectory tracking ability of a perturbed robot arm manipulator.

In [46], the authors applied VEPSO to the problem of determining the generator contributions to a transmission system. The authors used a multi-objective optimization approach to model the problem of evaluating the generator contributions. The VEPSO was applied using a network of processors. Although the approach was slower than the analytical methods, it was found to produce accurate results when compared to them while considering the nonlinear characteristics of the system as well.

2.3 Classifying the Cooperative PSO Models

The taxonomy proposed in [47] is extended and used to cover the different implementations surveyed. This taxonomy is shown in Figure 2.6.

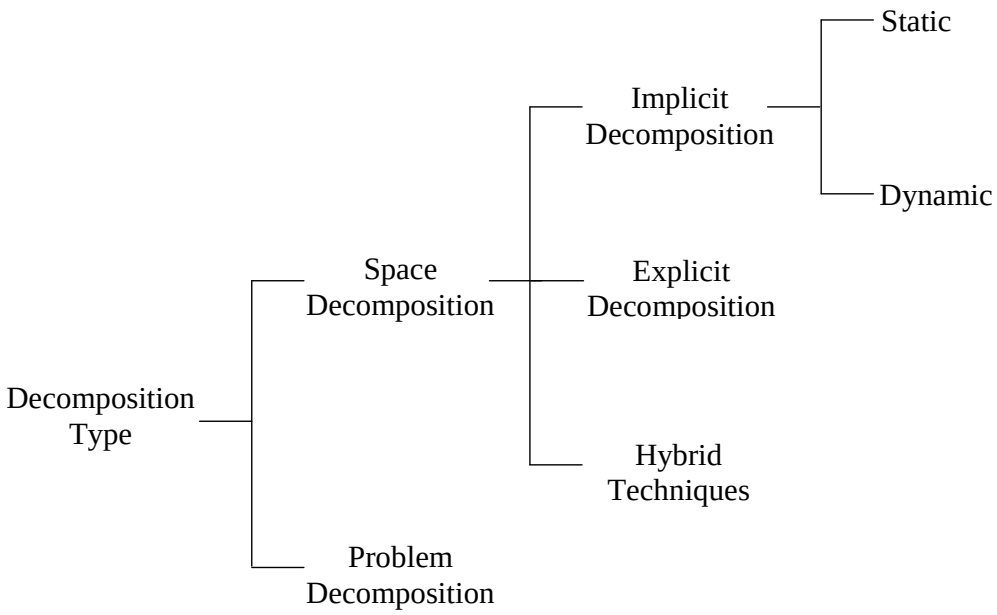


Figure 2.6: Decomposition-Type based taxonomy.

In *problem decomposition*, the problem itself is divided into several sub-problems, each one

is solved using a different swarm. The swarms share the solutions they find with each other in order to reach a global solution for the problem in hand. This class includes the implementations in [35–37, 46] as each swarm is optimizing a single objective function, [40, 41, 45] as each swarm is solving a different part of the problem (min and max parts) and [44] as each swarm is evolving for a different player.

The *implicit space decomposition*, where complete solutions are being shared, involves the decomposition of the search space between different swarms. The name implicit comes from the fact that the different swarms explore different areas in the search space due to different initial solutions, different parameter settings or both. It also comes from the fact that the swarms may follow different *gbests* in different regions. This class is further divided into *static* swarms and *dynamic* swarms. The *static* swarms class involves implementations where the swarms are separate, swarm do not get re-constructed, no introduction of new swarms and no deletion of existing swarms while the model is running; this class includes the implementations in [10, 14, 15, 20, 27, 28, 34, 38, 39, 43]. The *dynamic* swarms class includes the implementations involving the continuous re-construction of existing swarms [21, 22, 25, 26, 29, 31–33], the introduction of new swarms or the deletion of old swarms as in [23].

In the *explicit space decomposition*, where partial solutions are being shared, the search space is explicitly decomposed into sub-spaces. Each swarm searches for a sub-solution in a different sub-space of the problem. Hence, each swarm provides a partial solution to the problem, these partial solutions are gathered to provide the complete solution. The implementation falling under this class is the CPSO reported in [9, 11] and its application to neural network training reported in [42].

The *hybrid* approach refers to the idea of having a cooperative system that employs both methods of space decomposition. This class includes the CPSO_H reported in [9] and the hierarchal cooperative PSO in [16].

2.4 Conclusion

This chapter surveys the different cooperative PSO implementations proposed in the literature. All the different implementations are categorized according to the application they were used in. A taxonomy to classify all the surveyed models is proposed. This taxonomy classifies the cooperative models based on two different aspects: the decomposition approach adopted by the model and the method used for placing the particles into the different cooperating swarms. The way the search

space is decomposed is done by either decomposing the search space or decomposing the problem in hand. While the particles gets placed in the cooperating swarm either statically or dynamically. The taxonomy helps in gathering all the proposed models under one roof and understanding the similarities and differences between these models.

Chapter 3

Information Exchange in Cooperative Particle Swarm Optimizers

This chapter starts by giving a definition of what is meant by a cooperative PSO algorithm based on the models surveyed in the previous chapter. The given definition helps in identifying key design decisions (parameters settings) that affect the performance of any cooperative PSO algorithm. These decisions mainly give answers to four important questions; Which information to share? When to share it? Whom to share it with? and What to do with it? The chapter summarizes the design decisions taken by the different surveyed models. A complete empirical study is then carried on one cooperative PSO model to observe how the performance is influenced by these parameters.

3.1 Definition

Based on the different cooperative PSO models surveyed we can define a cooperative PSO model as follows: Multiple swarms (or sub-swarms) searching for a solution (serially or in parallel) and exchanging some *information* during the search according to some *communication strategy*. Based on the exchanged *information*, an *action* is taken to effectively continue with the search process.

From the proposed definition, when implementing a cooperative PSO model, one has to decide upon several design issues in order to get the best performance. These issues are related to the following:

- **Which** information to exchange? Most of the cooperative PSO models up to date rely on exchanging the best experience of the cooperating swarms, referred to as *gbest*, the best particle, the attractors, or the leaders.

- **When** to exchange the information? To answer this question we should decide upon a *communication strategy* to use. This could be either *synchronous* or *asynchronous*. In synchronous communication, the cooperating modules exchange information with each other every predetermined number of iterations. On the other hand, asynchronous communication involves the exchange of information when a certain condition occurs (*e.g.*, when the solution found by a certain module does not improve for a specified number of iterations).

- **Whom** to share this information with? In other words, does the communication only occur between neighboring swarms or do all the swarms share the information among each other?

- **What** to do with the exchanged information? This defines the approach taken by any swarm to deal with the received information. Many actions have been proposed to handle this information including replacing some of its own particles, using it to update its particles velocities, and re-initializing the whole swarm.

These issues are similar to the ones raised in [48], while adding the fourth issue.

Tables 3.1, 3.2 and 3.3 summarize the cooperative PSO models surveyed in the previous chapter highlighting their choice of exchanged information, communication strategies ('S' Synchronous or 'A' Asynchronous), when the communication occurs (number of iterations or required condition), and the actions taken after the communication is carried out (based on the exchanged information).

Tables 3.1 and 3.2 show that the *static* implementations of the model, where the swarms are fixed, always relies on exchanging the best particle (or list of particles) among the cooperating swarms and usually adopting the synchronous type of communication. The only exception is in [28,34], where the asynchronous communication is partially used. While in the *dynamic* implementations shown in Table 3.3, since the swarms get continuously reconstructed, the information exchange step involves all the particles. It is also interesting to note that both the Niche_PSO and the Multi_Swarm approaches check whether the swarms are close to each other after every iteration. However, they operate differently, the first approach merges the close swarms and the second one re-initializes the swarm following the bad attractor.

3.2 The Which, When, Whom and What Parameters

In order to study the effect of the different choices that could be adopted, a general PSO cooperative model is taken as a test case. This cooperative model is similar to CONPSO with both swarms performing the *gbest* model. The model adopts the sequential algorithm shown in Algorithm

Table 3.1: Static cooperative PSO models.

Model	**Which**	**Comm. strategy**	**When**	**What**
CPSO_S [9, 11, 42]	Best Particle	S	Every iteration	Updates the context vector
CPSO_H [11]	Best Particle	S	Every iteration	Replaces a random particle
CONPSO [10]	*gbest*	S	Not specified	Follow the better *gbest*
PPSO [14, 15]	*gbest*	S	Every 20 iterations	Mutate and replace poor particles
Diversity-based [17]	List of particles	S	Every 10 iterations	Replaces a list of particles
MCPSO [20]	*gbest*	S	Every iteration	Add to velocity update equation
MGPSO [27]	*gbest*	S	Every iteration	Repulsion in velocity if necessary different *gbest* update
Multi-Swarms I [28]	Attractors	S A	Every iteration Close Attractors	Are attractors close? Re-initialize swarm with bad attractor
Multi-Swarms II [34]	Attractors	S A A	Every iteration Close Attractors Swarms converged?	Are attractors close? Re-initialize bad attractor swarm Re-initialize the worst swarm
VEPSO [35, 36, 46]	*gbest*	S	Every iteration	Add to velocity update equation
MSPSO [37]	*gbest*	S	Every iteration	Add to velocity update equation

3.1 using two cooperating swarms while adopting synchronous communication. The information exchanged is the global best of the two swarms and the action taken is that both swarms follow the better *gbest*.

It is necessary to have both swarms use the same PSO algorithm for studying the number of swarms parameter discussed in this section. If the swarms are using different algorithms, we would be facing the problem of which swarm to choose to add to the cooperative system.

Table 3.2: Static cooperative PSO models, contd.

Model	Which	Comm. strategy	When	What
AMOPSO [38]	Swarm's leaders	S	Every 5 iterations	Re-assign leaders to swarms
CMOPSO [39]	*guide*	S	Every 20 iterations	Re-initialize sub-swarm in $guide's$ neighborhood
Co-evolving Swarms [40, 41, 45]	All particles	S	Every 10 iterations	Information used in fitness evaluation
Swarm Music [43]	Targets	S	Every iteration	Add to velocity update equation
		A	Swarms converged?	Re-initialize the worst swarm

Table 3.3: Dynamic cooperative PSO models.

Model	Which	Comm. strategy	When	What
Niche PSO [23]	All particles	S	Every iteration	Swarms close?
		A	Close Swarms	Merge Swarms
		A	No fitness change	Create new swarms
DMS-PSO [21, 22]	All particles	S	Every 3 iterations	Re-construct sub-swarms randomly
PH-PSO [29, 31]	All particles	A	A change occurs	Splitting hierarchy
		S	10 iterations	Re-joining hierachy
PH-PSO-a [29, 31]	All particles	A	A change occurs	Splitting hierarchy
		A	A swarm contains *gbest* for 5 iterations	Re-joining hierarchy
SPSO [25, 26, 32, 33]	All particles	S	Every iteration	Re-construct the swarms

3.2.1 Which Information to Share? and What to do with it?

In the CONPSO model, the two swarms exchange their global bests, a choice usually taken by static implementations. The swarms then follow the better *gbest*. In this case, the flow of information is only from one swarm to the other. The swarm that has the better global best provides new information to the other swarm without gaining anything. To overcome this disadvantage, another information sharing mechanism should be adopted. This requires changing the action taken after the communication step or choosing different information to exchange.

Algorithm 3.1 A cooperative sequential algorithm.

Require: Max_Function_Evaluations

1: Initialize the two swarms
2: Max_Iterations = $\frac{Max\_Function\_Evaluations}{Num\_Particles}$
3: iter_number = 1
4: **while** $iter\_number \leq Max\_Iterations$ **do**
5: Update swarm 1
6: Update swarm 2
7: **if** Synchronization **then**
8: Exchange Information()
9: **end if**
10: iter_number = iter_number + 1
11: **end while**
12: **return** min($gbest_1$,$gbest_2$)

The action taken is changed by allowing each swarm to receive the coming *gbest* and use it to replace one of its particles. The particle replaced could be either the worst particle or a randomly chosen particle. The same thing applies for choosing the information to be sent to the other swarms. One might choose to send the best particle or even send a randomly chosen particle.

This gives rise to four different information exchange approaches that are tested in this work. The general information sharing mechanism is carried out by selecting a particle from one swarm (the information chosen) replacing the contents of another particle in another swarm (the action taken). Different information exchange mechanisms could be adopted by choosing the way a certain particle is selected or replaced. Theses approaches are:

- Selecting the best particle to replace the worst one (Best-Worst),
- Selecting the best particle to replace a random one (Best-Random),
- Selecting a random particle to replace the worst one (Random-Worst),
- Selecting a random particle to replace a random one (Random-Random).

The method is implemented by replacing the *information exchange* step in Algorithm 3.1 by the steps shown in Algorithm 3.2, where N is the number of particles in the swarm and p is the number of exchanged particles. In the algorithm shown, one swarm chooses its best p particles to

replace the worst p particles in the other swarm. All four information exchange approaches are experimented with in this work and compared with just sharing the same global best.

Algorithm 3.2 Exchanging the particles information both ways.

```
1: Sort the particles inside each swarm
2: k = 1
3: while k ≤ p do
4:    Replace particle N-k in swarm 1 with particle k in swarm 2
5:    Replace particle N-k in swarm 2 with particle k in swarm 1
6:    k = k + 1
7: end while
```

It should be noted that there are still many ways available for sharing information. In [49], the authors experimented with performing the crossover operator between particles that belong to different sub-populations, which could be regarded as a different type of information exchange.

3.2.2 When to Share Information?

If synchronous communication is used, one has to set the number of iterations after which the modules exchange information. This parameter is referred to in this work as the *synchronization period*. This parameter is being tested while sharing the global best solution between the two cooperating swarms. Note also that a similar parameter could be identified in asynchronous communication, which is the number of iterations that a module has to wait for, before performing the exchange, provided that the solution quality does not improve.

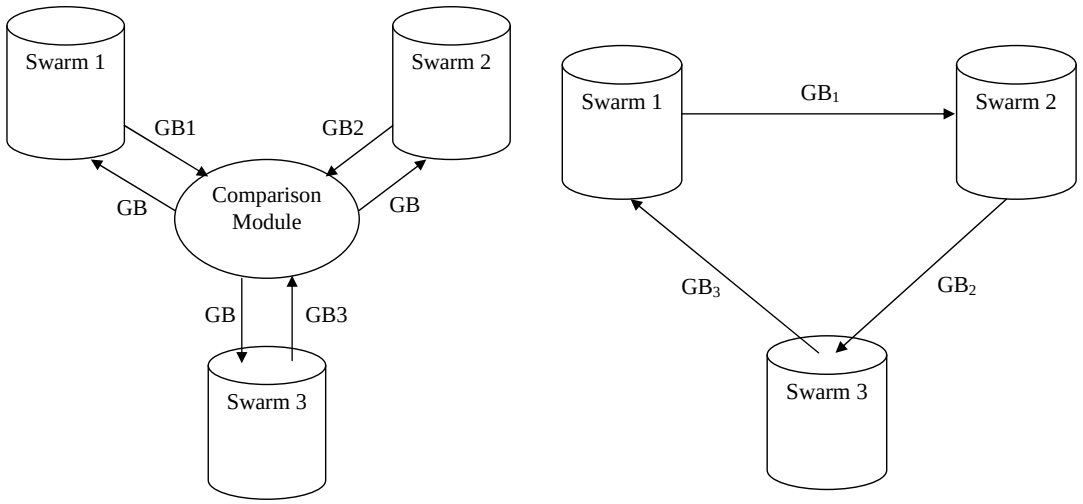
When having a single swarm with $2n$ particles, these particles are used to update the *gbest* value of the swarm after every iteration. Assuming that we have two swarms with n particles each, and that each swarm performs T iterations. We wish to investigate how the performance of the two swarms model changes while changing the synchronization period t_{sync}. If t_{sync} is equal to one, the two swarms will share their global bests after every iteration, hence, we may find that the performance of the two swarms model is statistically equivalent to the performance of a single swarm having $2n$ particles. On the other hand, if $t_{sync} > T$, the two swarms will never communicate during their search and the model becomes equivalent to two independent runs of a single swarm with n particles. It is still unclear how would the cooperative model behave if $1 < t_{sync} < T$.

3.2.3 Whom to Share Information With?

This question is answered by defining the neighborhood topology of the cooperating swarms. Two different neighborhood topologies are tested in this work:

- Fully-connected topology: Sharing the global minimum among all swarms,
- Ring topology: Circular communication of the global minimum in a directed ring fashion.

Figure 3.1 shows both topologies, where GB_i refers to the global best of swarm i and GB is the minimum of all global bests.



(a) Fully-connected topology - global sharing (b) Ring topology - circular communication

Figure 3.1: Different neighborhood topologies.

Again, this parameter is being tested while sharing the global best among all the cooperating swarms. To test the ring topology, the *information exchange* step in Algorithm 3.1 is replaced by the steps shown in Algorithm 3.3.

3.2.4 Additional Parameters

- Number of cooperative modules: In PSO, if the number of particles is kept fixed, increasing the number of cooperating swarms decreases the number of particles per swarm, which could affect the performance of the cooperative model. Previous experiments conducted in [28], [42], [35] that studied changing the number of swarms, all reached a similar conclusion.

Algorithm 3.3 The information exchange step when adopting the ring topology.

1: **for** each swarm s **do**
2: Report $gbest_s$ to swarm s+1
3: **end for**
4: **for** each swarm s **do**
5: $gbest_s = min(gbest_s, gbest_{s-1})$
6: **end for**

Increasing the number of swarms helps to improve the performance up to a certain limit. However, in all these experiments, the swarms exchange information after every iteration. The experiments provided in this thesis are different because the synchronization period and the number of swarms used are both changing,

- Implementation approach: As stated in [50], cooperative algorithms are still efficient even if they are sequentially implemented. However, it is interesting to see whether the implementation approach taken to build the model (serial or parallel) affects the performance. In [35], the authors stated that the parallel implementation helped in reducing the execution time, however, they did not refer to the solution quality obtained, which suggests that it was not affected. A similar conclusion was also drawn in [51].

3.3 Results and Discussions

3.3.1 Experimental Settings

The experiments are run using the benchmark functions shown in Table 3.4. The *gbest* model is applied by decreasing w linearly from 0.9 to 0.1 and setting both $c1$ and $c2$ to 2. For all experiments, all the particles have been randomly initialized in the specified domain using uniform distribution. The experiments are applied to a problem dimensionality of 10, while performing 100000 function evaluations. The results reported are the averages taken over 50.

3.3.2 How are the experiments divided?

Due to the complexity of the experiments and the different number of parameters tested. A brief introduction is given to show how the experiments will be conducted.

Table 3.4: Benchmark functions.

Function	**Equation**	**Domain**	**Property**
Spherical	$f(x) = \sum_{i=1}^{n} x_i^2$	100	unimodal
Quadric	$f(x) = \sum_{i=1}^{n} \left(\sum_{j=1}^{i} x_j^2 \right)^2$	100	unimodal
Rosenbrock	$f(x) = \sum_{i=1}^{n/2} \left(100(x_{2i} - x_{2i-1}^2)^2 + (1 - x_{2i-1})^2 \right)$	2.048	unimodal
Griewank	$f(x) = \frac{1}{4000} \sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$	600	multimodal
Ackley	$f(x) = 20 + e - 20 \exp\left(-0.2 \sqrt{\frac{1}{n} \sum_{i=1}^{n} x_i^2} \right)$ $- \exp\left(\frac{1}{n} \sum_{i=1}^{n} \cos 2\pi x_i\right)$	30	multimodal
Rastrigin	$f(x) = \sum_{i=1}^{n} \left(x_i^2 - 10 \cos 2\pi x_i + 10 \right)$	5.12	multimodal
Schwefel	$f(x) = 418.9829 * n + \sum_{i=1}^{n} -x_i \sin \sqrt{\lvert(x_i)\rvert}$	500	multimodal

Firstly, a single swarm algorithm will be applied to all the functions with an increasing number of particles. The aim is to find out the best result achieved by a single swarm for each function as this will be used as a basis of comparison.

Secondly, the cooperative model is applied using 2 swarms and 10 particles per swarm, which are arbitrary chosen values. The experiments will be repeated while increasing the synchronization period allowing different number of communication steps. The experiments aim at finding out how the model behaves while changing this parameter and how good is the performance of the cooperative model compared to the single swarm.

Thirdly, the cooperative model is applied using three swarms and 10 particles per swarm. The experiments will be run using the two neighborhood topologies identified in Section 4.3 across different synchronization periods. The aim is to find out which of these topologies is better.

Fourthly, the cooperative model is applied while changing the number of swarms with a fixed

number of particles. The goal is to find a suitable number of swarms to be used and how would changing this parameter affect the synchronization period.

Finally, the experiments are repeated for a higher dimensionality to check whether the reached conclusions still hold.

3.3.3 Results of A Single Swarm

Tables 3.5, 3.6, and 3.7 show the results obtained using a single swarm and performing the same number of function evaluations while changing the number of particles. The results generally indicate that the solution improves by increasing the number of particles up to a certain limit, then the quality of the results tends to deteriorate. Except for the Ackley and the Schwefel functions where the results still improve.

Table 3.5: Results of a single swarm for the unimodal functions for a dimensionality of 10.

Number of	**Spherical**		**Quadratic**		**Rosenbrock**	
Particles	**Mean**	**Std.**	**Mean**	**Std.**	**Mean**	**Std.**
10	2.702e-01	3.700e-01	1.930e+00	2.422e+00	3.147e-01	3.136e-01
20	4.535e-03	1.067e-02	4.814e-02	1.241e-01	6.663e-02	1.258e-01
30	2.849e-05	6.889e-05	3.190e-05	9.683e-05	3.501e-03	1.160e-02
50	8.807e-13	5.747e-12	2.887e-17	1.804e-16	3.835e-05	2.712e-04
100	9.965e-31	6.982e-30	1.224e-50	7.938e-50	8.751e-32	1.590e-31
200	**1.495e-47**	**4.527e-47**	**1.746e-55**	**1.171e-54**	**1.652e-32**	**2.422e-32**
300	2.150e-41	7.758e-41	6.738e-44	1.579e-43	2.465e-32	4.211e-32

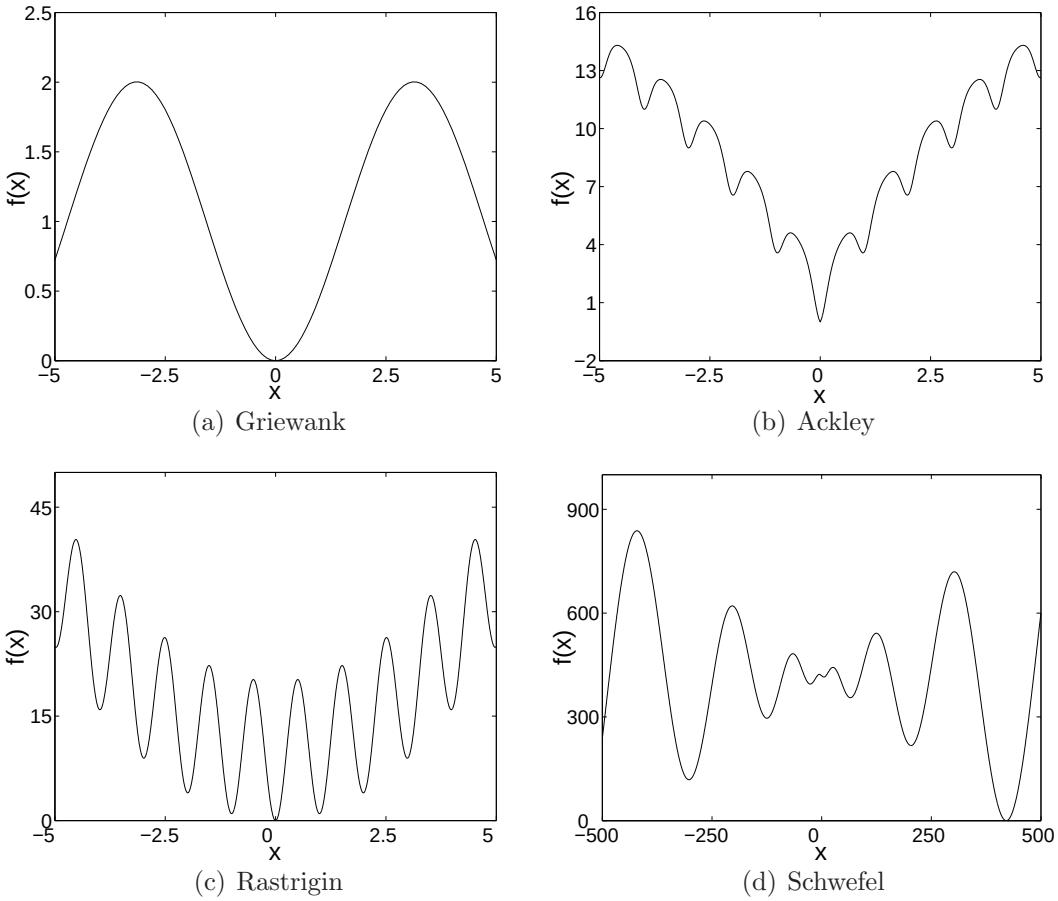
Table 3.6: Results of a single swarm for the multimodal functions for a dimensionality of 10.

Number of	**Griewank**		**Ackley**	
Particles	**Mean**	**Std.**	**Mean**	**Std.**
10	3.295e-01	1.292e-01	1.532e+00	9.033e-01
20	2.136e-01	1.185e-01	1.456e+00	9.340e-01
30	1.908e-01	1.255e-01	1.238e+00	9.203e-01
50	2.075e-01	1.148e-01	1.143e+00	8.681e-01
100	**1.594e-01**	**9.016e-02**	1.103e+00	1.038e+00
200	1.670e-01	1.159e-01	7.787e-01	8.512e-01
300	1.702e-01	1.165e-01	**6.423e-01**	**8.250e-01**

Table 3.7: Results of a single swarm for the multimodal functions for a dimensionality of 10, contd.

Number of Particles	**Rastrigin**		**Schwefel**	
	Mean	**Std.**	**Mean**	**Std.**
10	8.289e+00	2.883e+00	9.121e+02	4.175e+02
20	8.433e+00	3.769e+00	8.877e+02	3.767e+02
30	7.125e+00	2.592e+00	9.150e+02	3.691e+02
50	8.179e+00	2.990e+00	8.061e+02	3.512e+02
100	7.920e+00	2.864e+00	7.312e+02	2.528e+02
200	**7.048e+00**	**2.603e+00**	7.434e+02	3.101e+02
300	7.403e+00	2.622e+00	**6.905e+02**	**3.248e+02**

For the multimodal functions, the results show that PSO is most successful in solving the Griewank function, followed by the Ackley function, the Rastrigin function, and finally the Schwefel function. The reason for this is realized by inspecting Figure 3.2, which shows the topology of all these functions in each dimension.



(a) Griewank (b) Ackley

(c) Rastrigin (d) Schwefel

Figure 3.2: Each dimension of the multi-modal functions.

The figure shows that the first three functions share the property of having a periodic sequence of local minima. The Ackley and the Rastrigin functions have more local minima than the Griewank function over the same interval . The surface of the Rastrigin function is more rugged than that of the Ackley function. The Schwefel function has large peaks and basins and characterized by having the global optimum far from the next best solution.

The experiments are also run using the benchmark functions f6-f14 shown in Table 3.8 that were proposed in CEC2005 and available at [52]. In order to constrain the particles movement within the specified domain, any violating particle gets its position randomly re-initialized inside the specified domain. The results reported for these functions are the error values $f(x) - f(x*)$, where $x*$ is the global optimum, and are the averages taken over 25 runs.

Table 3.8: CEC05 benchmark functions.

Benchmark Function	**Description**	**Lower Domain**	**Upper Domain**
f6	shifted Rosenbrock	-100	100
f7	shifted rotated Griewank without bounds	0	600
f8	shifted rotated Ackley	-32	32
f9	shifted Rastrigin	-5	5
f10	shifted rotated Rastrigin	-5	5
f11	shifted rotated Weierstrass	-0.5	0.5
f12	Schwefel	-100	100
f13	expanded extended Griewank plus Rosenbrock	-3	1
f14	shifted rotated expanded Scaffer	-100	100

Tables 3.9, 3.10, and 3.11 show the results of applying the *gbest* model to these functions. The results show that increasing the number of particles up to a certain limit improves the performance as was shown before. The only exception is in f6, where having 10 particles provided the best solution and in f8 where changing the number of particles did not have a significant effect.

In the following experiments, the results of the cooperative model is compared to a single swarm performing the *gbest* model to observe how its performance is changing under the tested factor with respect to the single swarm performance. The results will not be presented for f8 as this function has the optimum on the bounds and the topology is similar to the needle-in-hay-stack problem, the cooperative model did not provide any improvements.

Table 3.9: Results of a single swarm for the CEC05 functions for a dimensionality of 10.

Number of Particles	f6		f7		f9	
	Mean	Std.	Mean	Std.	Mean	Std.
10	**3.158e+00**	**5.593e+00**	3.172e-01	1.678e-01	3.423e+00	2.053e+00
20	6.983e+00	1.592e+00	2.318e-01	1.462e-01	2.905e+00	1.882e+00
30	7.980e+00	1.041e+01	2.236e-01	1.858e-01	3.025e+00	1.268e+00
50	1.742e+01	3.874e+01	1.946e-01	9.274e-02	2.348e+00	1.344e+00
100	1.742e+01	3.874e+00	**1.403e-1**	**6.087e-2**	2.109e+00	1.124e+00
200	1.823e+01	3.378e+01	1.864e-01	9.808e-02	**1.395e+00**	**9.499e-01**
300	1.940e+01	3.273e+01	1.734e-01	7.336e-02	1.672e+00	1.243e+00

Table 3.10: Results of a single swarm for the CEC05 functions for a dimensionality of 10, contd.

Number of Particles	f10		f11		f12	
	Mean	Std.	Mean	Std.	Mean	Std.
10	1.541e+01	6.025e+00	4.128e+00	1.433e+00	6.073e+03	4.767e+03
20	1.577e+01	6.271e+00	4.242e+00	1.307e+00	6.387e+03	6.341e+03
30	1.445e+01	6.882e+00	**3.864e+00**	**1.434e+00**	**5.109e+03**	**3.554e+03**
50	1.389e+01	6.302e+00	4.294e+00	1.186e+00	6.417e+03	5.346e+03
100	1.317e+01	6.437e+00	4.456e+00	1.528e+00	6.671e+03	5.256e+03
200	**1.166e+01**	**5.469e+00**	4.553e+00	1.778e+00	9.765e+03	7.233e+03
300	1.174e+01	5.532e+00	4.701e+00	1.341e+00	1.032e+04	6.912e+03

Table 3.11: Results of a single swarm for the CEC05 functions for a dimensionality of 10, contd.

Number of Particles	f13		f14	
	Mean	Std.	Mean	Std.
10	6.758e-01	2.171e-01	2.725e+00	4.440e-01
20	6.227e-01	2.000e-01	2.872e+00	3.896e-01
30	6.305e-01	2.358e-01	2.690e+00	4.748e-01
50	6.442e-01	1.622e-01	**2.648e+00**	**4.305e-01**
100	5.615e-01	2.171e-01	2.949e+00	4.421e-01
200	**5.582e-01**	**1.798e-1**	2.920e+00	4.215e-01
300	6.728e-01	1.920e-01	2.999e+00	4.207e-01

3.3.4 Synchronization Period

In order to keep the number of function evaluations fixed, each swarm performed 50000 function evaluations when the cooperative model is applied. Hence, if there is any improvement in the

results, it should be due to the cooperative nature of the model. The model is re-run with different synchronization periods and the best results achieved by the cooperative model for all the functions are shown in Table 3.12. The results shown are the averages obtained over the 50 runs. The column titled "Compared to Single Swarm" is based on a two sample t-test that is run to verify the results statistical significance, where the null hypothesis is rejected with a 95% confidence level.

Table 3.12: Results of the cooperative model for a dimensionality of 10.

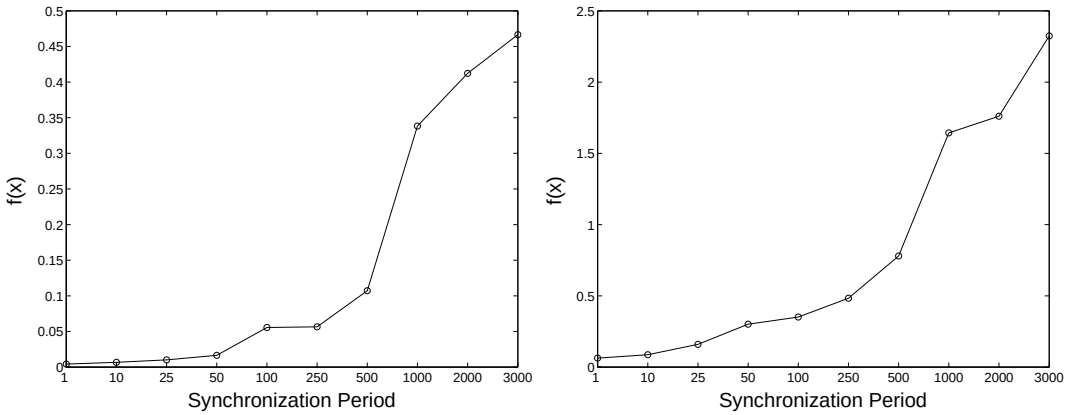
Benchmark Function	**Synchronization Period**	**Mean**	**Std.**	**Compared to Single Swarm**
Spherical	1	4.193e-03	8.525e-03	No improvement
Quadratic	1	6.329e-02	1.844e-01	No improvement
Rosenbrock	1	4.606e-02	7.997e-02	No improvement
Griewank	1	1.983e-01	1.683e-01	Similar to 50 particles
Ackley	1000	1.108e+00	7.386e-01	Similar to 100 particles
Rastrigin	2000	6.043e+00	2.548e+00	Better
Schwefel	500	6.661e+02	2.549e+02	Better

Figure 3.3 illustrates the model behavior for the unimodal functions for the different synchronization periods. The results show that reducing the synchronization period improves the solution quality. In fact, the results of the studied model approaches the result of the original PSO algorithm using 20 particles as the synchronization period decreases to 1, as discussed in Section 3.2. The same behavior is noticed for the Griewank function, which has sparse and small sized peaks causing it to behave like a unimodal one. That is why the single swarm is most successful in solving this function among all the multimodal ones.

For the other multimodal functions shown in Figure 3.4, increasing the synchronization period up to a certain limit produces better results because the probability of having the two swarms stuck at the same local minimum decreases. However, after a certain limit, the results start to deteriorate.

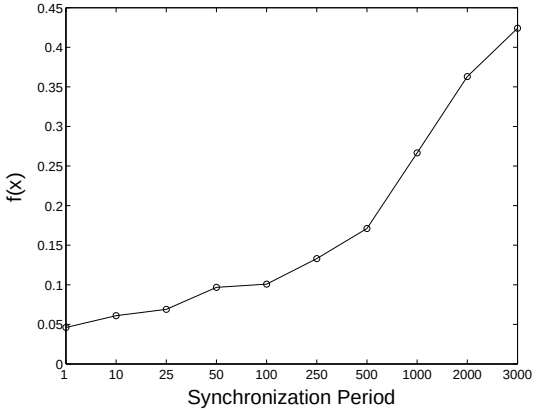
Since the Rastrigin function is more difficult than the Ackley function, the cooperative model needed a longer synchronization period, than what was needed for the Ackley function in order to produce the best results. This indicates that for harder functions more separation is needed.

A two sample t test is run to verify the results statistical significance. The test is run for all multimodal functions (except the Griewank function) between two samples drawn from the 50 runs. One sample is the output of the single swarm with 20 particles, and the other sample is



(a) Spherical

(b) Quadratic

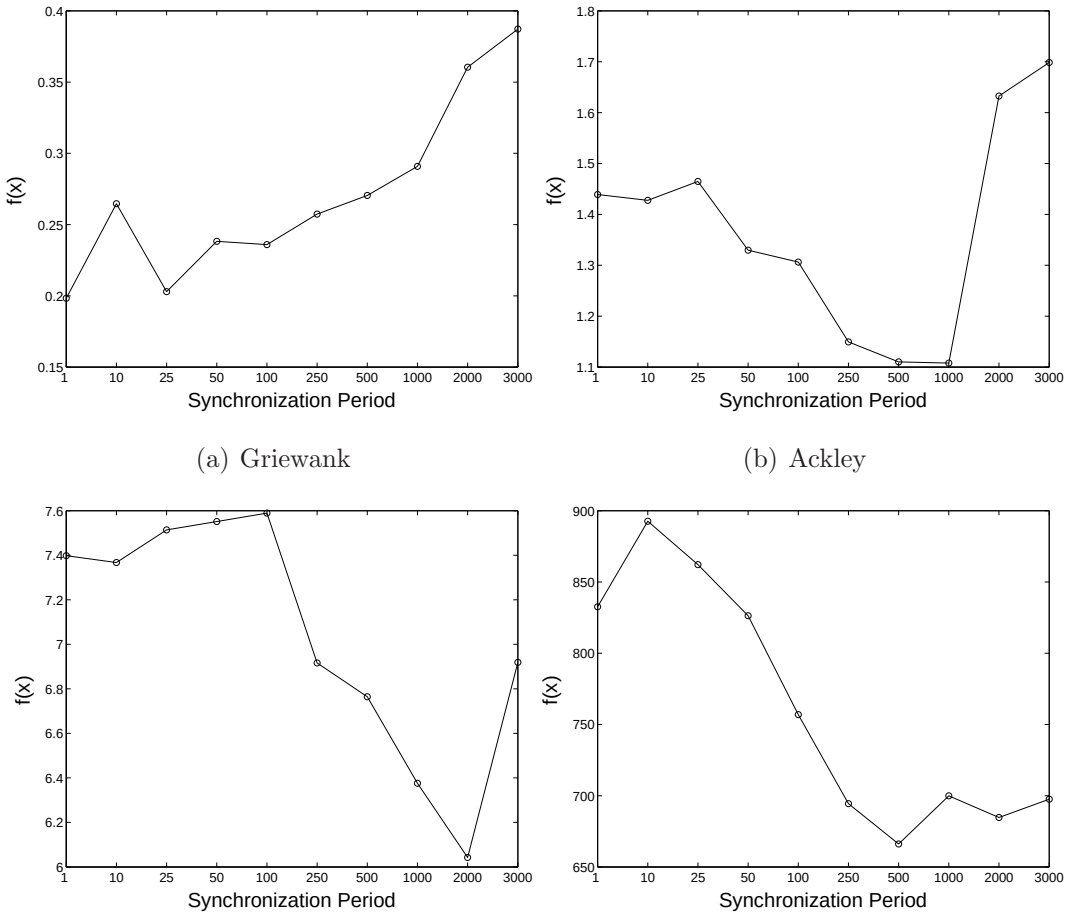


(c) Rosenbrock

Figure 3.3: Synchronization period effect for the unimodal functions for a dimensionality of 10.

the best output of the cooperative model. For all cases, the null hypothesis is rejected with a 95% confidence level indicating that adopting a 2-swarm cooperative model with n particles each is better than having a single swarm with $2n$ particles, provided that a suitable synchronization period is selected.

The experiments are rerun using the CEC05 functions to test if the conclusions reached still hold. The results in Table 3.13 marked with * show that the cooperative model was able to produce the best result achieved by a single swarm (often with a higher number of particles). Figure 3.5 and Figure 3.6 show that the cooperative model still has the same behavior as increasing the synchronization period improves the results.



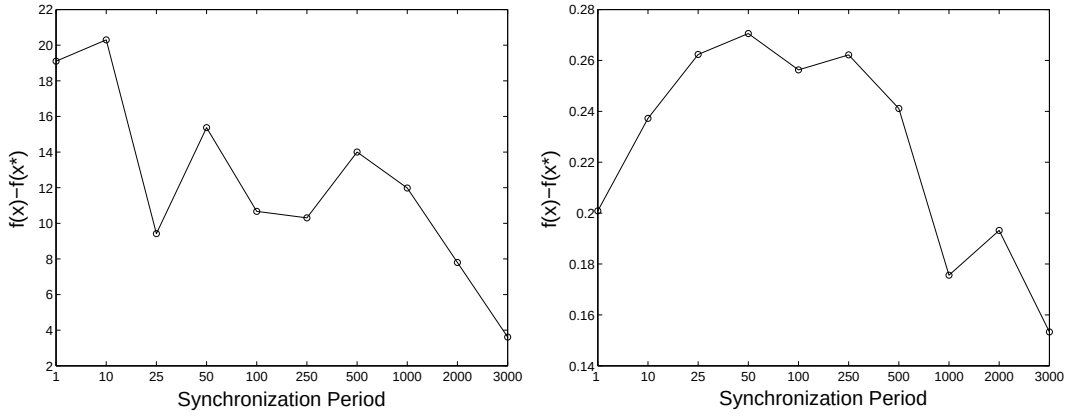
(a) Griewank (b) Ackley

(c) Rastrigin (d) Schwefel

Figure 3.4: Synchronization period effect for the multimodal functions for a dimensionality of 10.

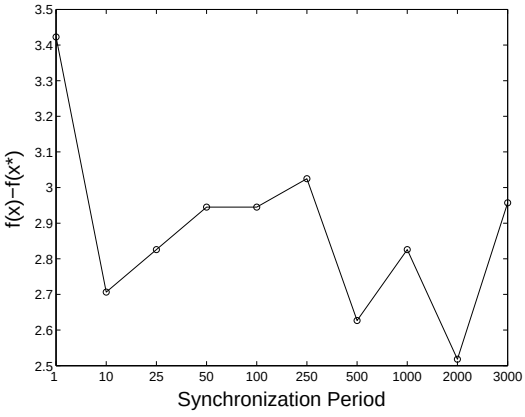
Table 3.13: Results of the cooperative model for the CEC05 functions for a dimensionality of 10.

Benchmark Function	**Synchronization Period**	**Mean**	**Std.**	**Against Single Swarm**
f6	3000	3.610e+00	5.393e+00	No improvement
f7	3000	1.533e-01	7.546e-02	Similar to 100 particles*
f9	2000	2.518e+00	1.796e+00	Similar to 100 particles
f10	500	1.300e+01	6.905e+00	Similar to 200 particles*
f11	2000	3.728e+00	1.123e+00	Similar to 30 particles*
f12	25	4.533e+03	3.999e+03	Similar to 30 particles*
f13	3000	5.672e-01	1.553e-01	Similar to 200 particles*
f14	100	2.819e+00	4.598e-01	Similar to 50 particles*

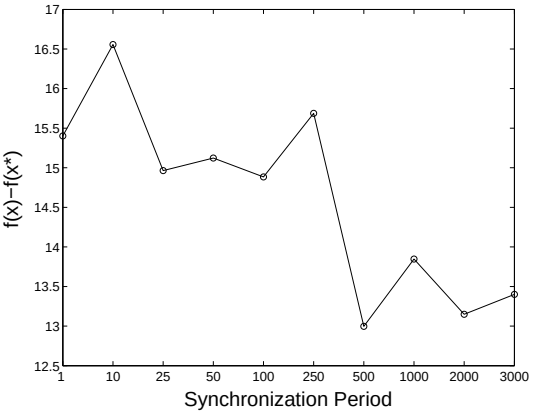


(a) f6

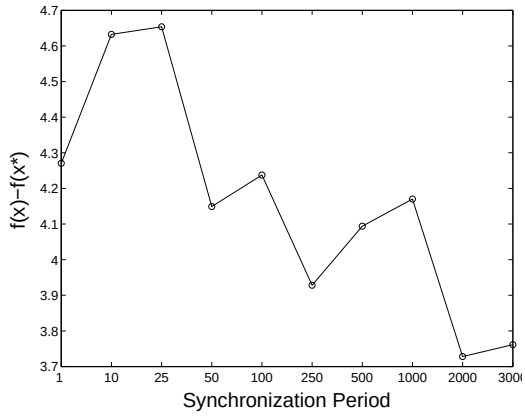
(b) f7



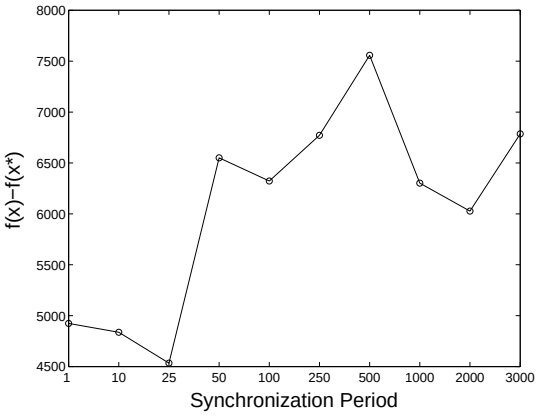
(c) f9



(d) f10



(e) f11

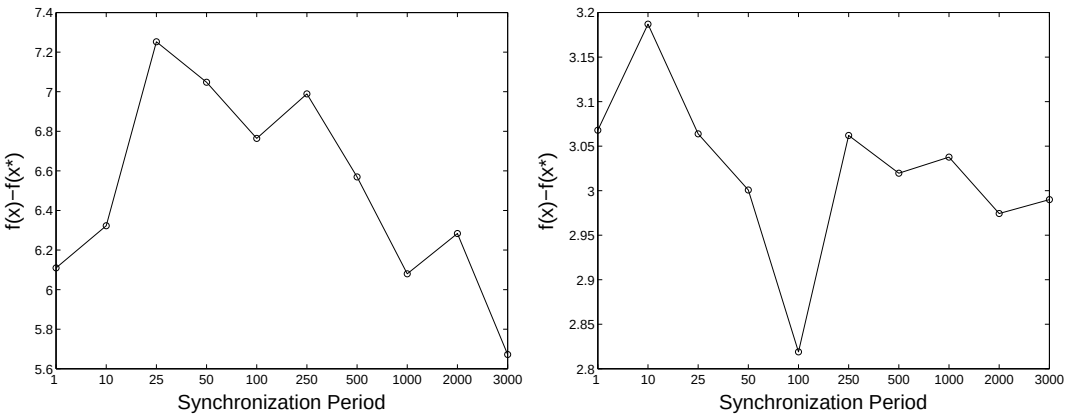


(f) f12

Figure 3.5: Synchronization period effect for the CEC05 functions.

3.3.5 Neighborhood Topology

This section experiments with the two neighborhood topologies previously identified by increasing the number of swarms to three.



(a) f13 (b) f14

Figure 3.6: Synchronization period effect for the CEC05 functions, contd.

Tables 3.14 and 3.15 show the best results obtained by applying both approaches and the results are also plotted in Figure 3.7. The results show that the circular communication approach gives better results than the global sharing approach because it maintains the diversity among the cooperating swarms. When global sharing is used, all the swarms follow the same global best solution after every synchronization point. This increases the probability of having all the swarms stuck at the same local minimum.

Table 3.14: Results of the global sharing approach for a dimensionality of 10.

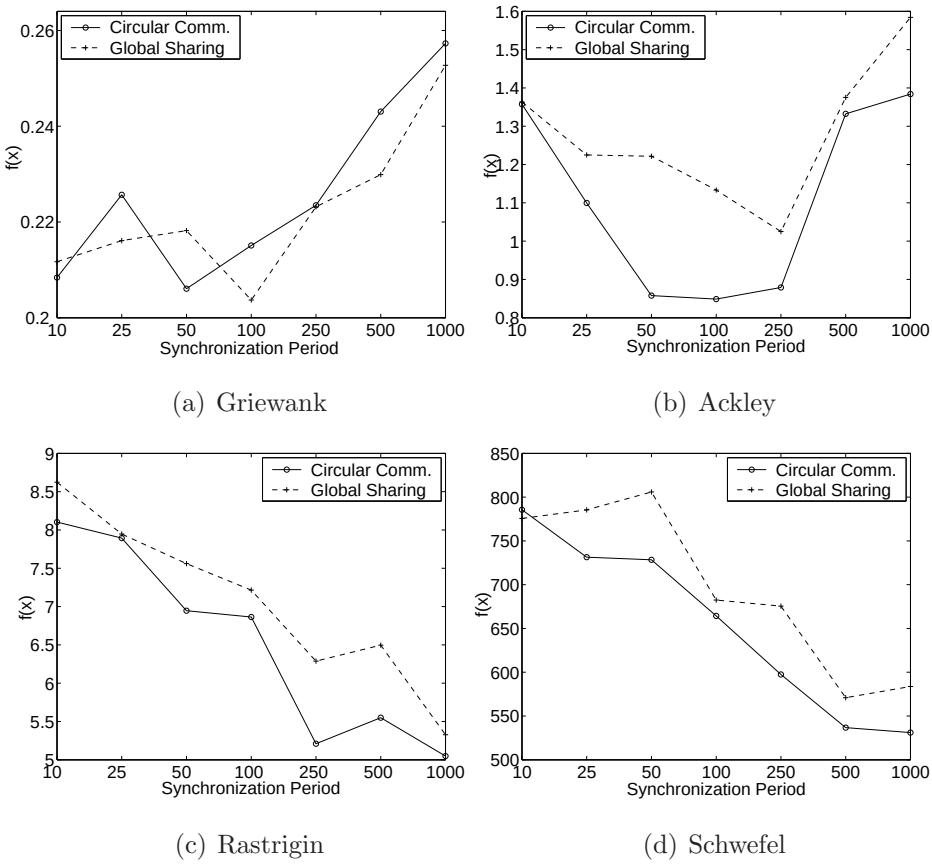
Benchmark Function	**Synchronization Period**	**Mean**	**Std.**	**Compared to Single Swarm**
Griewank	100	2.030e-01	1.272e-01	No Improvement
Ackley	250	1.025e+00	7.407e-01	Similar to 100 particles
Rastrigin	1000	5.327e+00	2.894e+00	Better
Schwefel	500	5.709e+02	2.455e+02	Better

Table 3.15: Results of the circular communication approach for a dimensionality of 10.

Benchmark Function	**Synchronization Period**	**Mean**	**Std.**	**Compared to Single Swarm**
Griewank	50	2.152e-01	1.329e-01	No Improvement
Ackley	100	8.488e-01	8.455e-01	Similar to 300 particles*
Rastrigin	1000	5.027e+00	2.282e+00	Better
Schwefel	1000	5.311e+02	2.093e+02	Better

For the Griewank function, the global sharing mechanism is sometimes better than the circular

communication approach behaving more like a unimodal function, when it comes to the cooperative model. The three swarms needed a longer synchronization period to produce the best result for the Rastrigin function than the value needed for the Ackley function. This emphasizes that more separation is needed to solve the Rastrigin function.



(a) Griewank (b) Ackley

(c) Rastrigin (d) Schwefel

Figure 3.7: Comparing two neighborhood topologies for a dimensionality of 10.

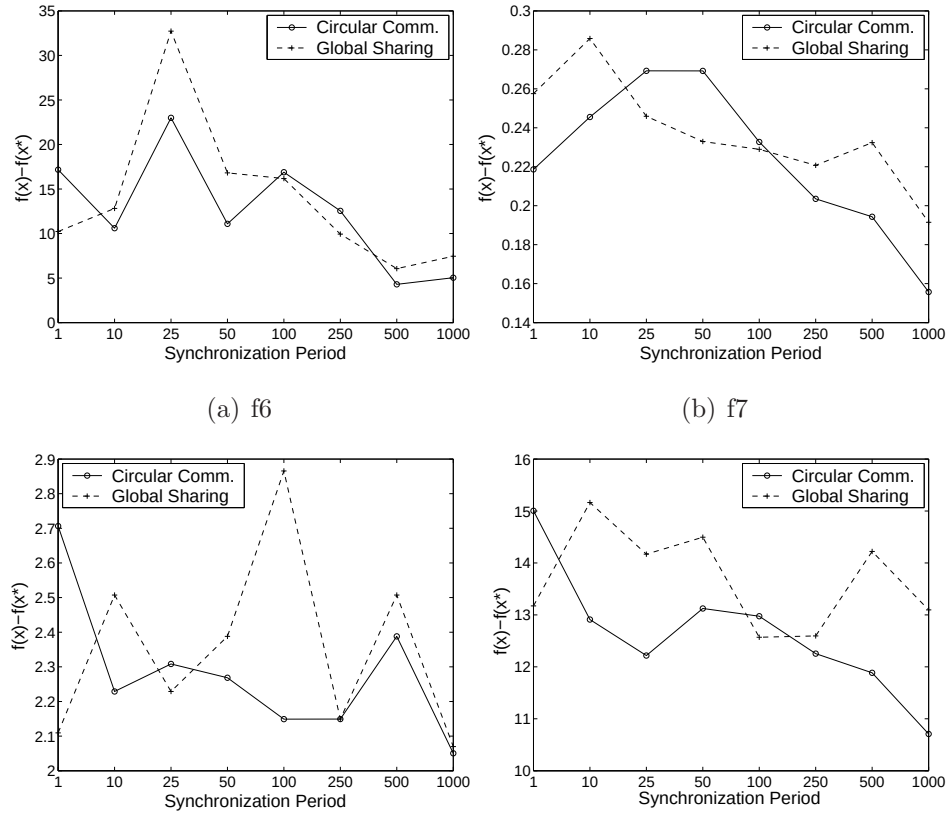
A two sample t test is run to verify the significance of the results. The two samples are the 50 runs giving the best overall result for a single swarm and the 50 runs giving the best result of the 3-swarms cooperative model. For all cases, the null hypothesis is rejected with a 95% confidence level. This means that adopting the 3-swarms technique outperformed the single swarm. The only exception is the Ackley function at a dimension equal to 10, where the first sample is the output of a single swarm with 100 particles (since the single swarm is still better with 300 particles).

Table 3.16 shows the best results obtained by applying the circular communication approach to f6-f14. Again, results marked with * show that the 3-swarms cooperative model was able to produce the best result achieved by a single swarm (often with a higher number of particles) for most of the functions. Figure 3.8 and Figure 3.9 illustrate the comparison between the two neigh-

borhood topologies, which shows that circular communication provides better results as previously concluded.

Table 3.16: Results of the circular communication approach for the CEC05 benchmark functions for a dimensionality of 10.

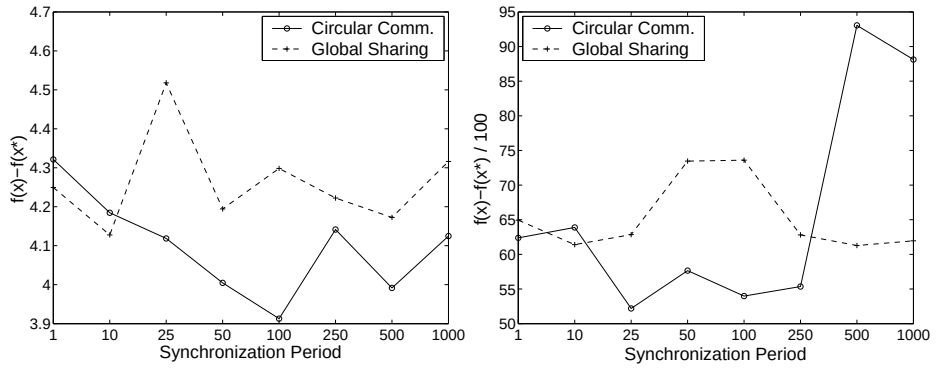
Benchmark Function	**Synchronization Period**	**Mean**	**Std.**	**Against Single Swarm**
f6	500	4.299e+00	4.852e+00	No Improvement
f7	1000	1.557e-01	9.502e-02	Similar to 100 particles*
f9	1000	2.050e+00	1.168e+00	Similar to 100 particles
f10	1000	1.071e+01	2.259e+00	Similar to 200 particles*
f11	100	3.913e+00	1.521e+00	Similar to 30 particles*
f12	25	5.219e+03	3.708e+03	Similar to 30 particles*
f13	25	5.635e-01	1.738e-01	Similar to 200 particles*
f14	250	2.614e+00	4.659e-01	Similar to 50 particles*



(a) f6 (b) f7

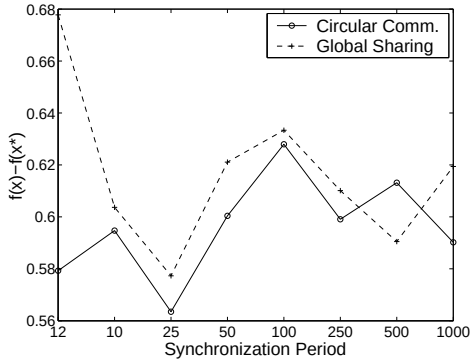
(c) f9 (d) f10

Figure 3.8: Comparing two communication strategies for the CEC05 functions for a dimensionality of 10.

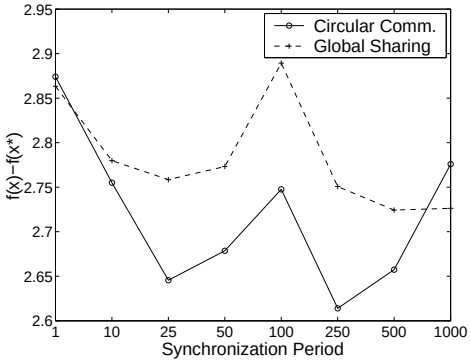


(a) f11

(b) f12



(c) f13



(d) f14

Figure 3.9: Comparing two communication strategies for the CEC05 functions for a dimensionality of 10, contd.

3.3.6 Number of Swarms

Changing the number of the cooperating swarms is investigated in this section. When comparing the performance of the cooperative PSO model with different number of swarms, the number of particles will be kept fixed. This means that increasing the number of cooperating swarms will decrease the number of particles per swarm. Hence, raising an important question, is it better to have a small number of swarms with a large number of particles per swarm? Or is it better to have many swarms with less number of particles in them?

To test the effect of increasing the number of swarms, experiments are run using 2, 3, 5, and 10 cooperating swarms adopting the circular communication approach. The number of particles is always kept fixed at 30. Table 3.17 shows the best results produced by every system and the synchronization period at which this result is produced.

The results show that if the number of swarms is increased, the best solution obtained will be

Table 3.17: Varying the number of swarms for the multimodal functions for a dimensionality of 10.

No. of	**Ackley**			**Rastrigin**			**Schwefel**		
Swarms	**S**	**Mean**	**Std.**	**S**	**Mean**	**Std.**	**S**	**Mean**	**Std.**
2	250	1.156e+00	8.468e-01	**1000**	**4.873e+00**	**1.774e+00**	3000	6.353e+02	3.001e+02
3	100	8.488e-01	8.455e-01	1000	5.027e+00	2.282e+00	1000	5.311e+02	2.093e+00
5	**50**	**7.580e-01**	**7.031e-01**	250	5.322e+00	1.866e+00	100	4.487e+02	2.059e+02
10	25	9.177e-01	6.975e-01	50	5.760e+00	1.808e+00	**100**	**4.339e+02**	**2.365e+02**

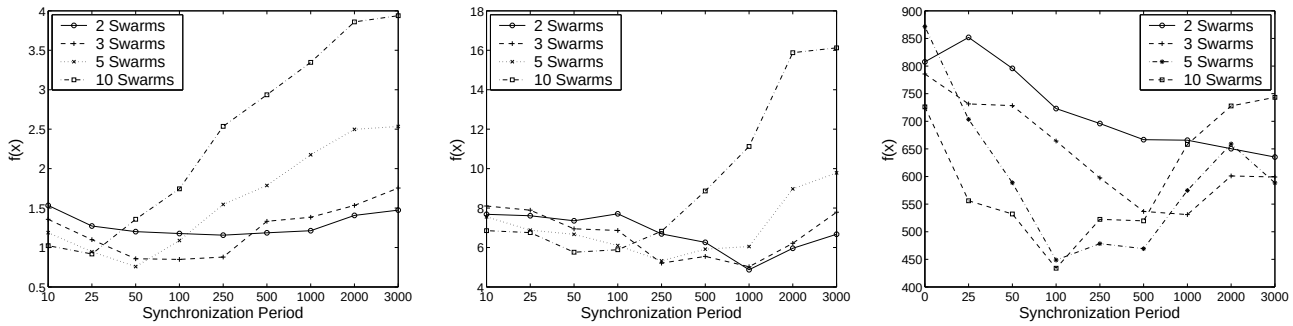
achieved at a shorter synchronization period than usual. This indicates that if more swarms are used, more communication is needed because the number of particles per swarm is small. Since the Rastrigin function is harder to solve than the Ackley function, the best results achieved are obtained at longer synchronization periods. For the Schwefel function, increasing the number of swarms up to 10 is useful, while having a shorter synchronization period. Due to the long distance between the two best minima in this function, having more swarms will increase the chance of having at least one of them reaching the global one.

The results show that increasing the number of swarms while having a long synchronization period is not beneficial. Increasing the number of swarms will decrease the number of particles per swarm, while having a longer synchronization period will keep the swarms more separated. This will make it difficult for the swarms to escape local minima.

Figure 3.10 shows a somehow similar behavior for the Ackley and the Rastrigin functions. For the Ackley function, increasing the number of swarms up to 5, helped to achieve better results provided that one uses the appropriate synchronization period. For the Rastrigin function, having two swarms only gave the best results. However, the synchronization period is rather long. It is also worth noting that since the Rastrigin function is harder to solve than the Ackley function, the best results achieved by any number of swarms are obtained at longer synchronization periods.

Tables 3.18, 3.19, and 3.20 show the results for the CEC05 benchmark functions. The results still show that increasing the number of swarms will usually limit the synchronization period.

Figure 3.11(a) shows the different balancing strategies that could be adopted between the number of swarms and the synchronization period. As the number of swarms decrease (increasing the number of particles per swarm) while decreasing the synchronization period as well, the particles of the different swarms become *tightly coupled*. One might reach the extreme case when both are set to one, adopting a single swarm approach. On the other hand, the particles become *loosely*



(a) Ackley (b) Rastrigin (c) Schwefel

Figure 3.10: Results of increasing the number of swarms.

Table 3.18: Varying the number of swarms for the CEC05 benchmark functions for a dimensionality of 10.

No. of	**f6**			**f7**			**f9**		
Swarms	**S**	**Mean**	**Std.**	**S**	**Mean**	**Std.**	**S**	**Mean**	**Std.**
2	**3000**	**2.122e+00**	**2.163e+00**	3000	1.592e-01	7.200e-02	1000	2.109e+00	1.417e+00
3	3000	2.204e+00	2.424e+00	1000	1.557e-01	9.502e-02	1000	2.050e+00	1.168e+00
5	2000	2.489e+00	3.640e+00	**500**	**1.272e-01**	**7.781e-02**	**250**	**1.810e+00**	**1.361e+00**
10	100	6.006e+00	5.202e+00	50	1.617e-01	8.953e-02	100	2.192e+00	1.351e+00

Table 3.19: Varying the number of swarms for the CEC05 benchmark functions for a dimensionality of 10, contd.

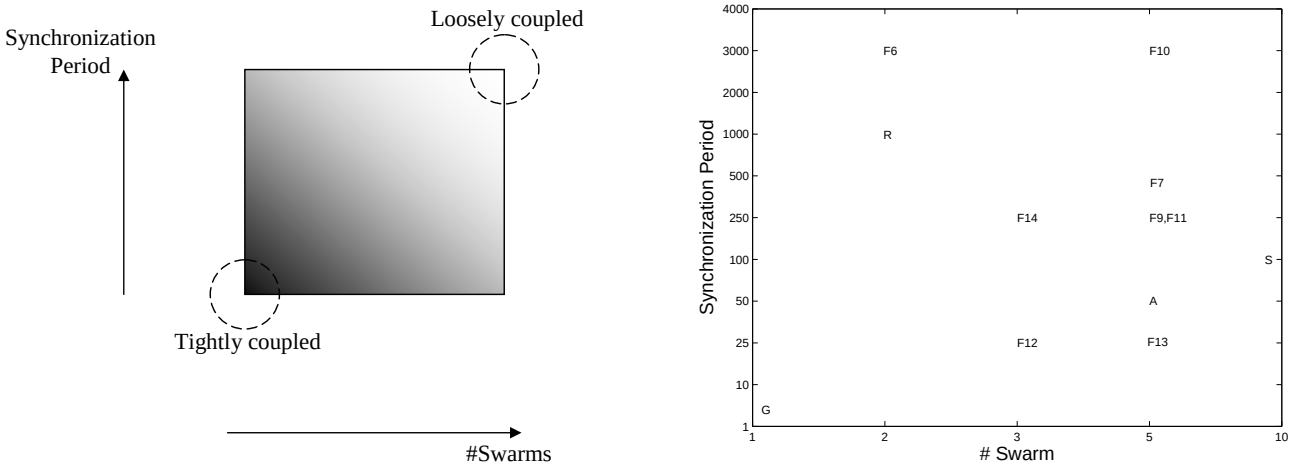
No. of	**f10**			**f11**			**f12**		
Swrms	**S**	**Mean**	**Std.**	**S**	**Mean**	**Std.**	**S**	**Mean**	**Std.**
2	3000	1.150e+01	5.107e+00	3000	3.944e+00	1.040e+00	25	5.582e+03	4.542e+03
3	1000	1.071e+01	2.259e+00	100	3.913e+00	1.521e+00	**25**	**5.219e+03**	**3.708e+03**
5	**3000**	**9.845e+00**	**4.200e+00**	**250**	**3.725e+00**	**1.246e+00**	25	5.397e+03	3.625e+03
10	2000	1.139e+01	3.740e+00	50	3.979e+00	1.420e+00	50	7.624e+03	4.127e+03

coupled when the number of swarms increase while increasing the synchronization period at the same time. At the extreme case, this may result in having totally independent particles (one particle per swarm and no communication).

To summarize the results, Figure 3.11(b) shows how each function is better optimized. The Griewank, the Ackley, the Rastrigin, and the Schwefel functions are better solved using different

Table 3.20: Varying the number of swarms for the CEC05 benchmark functions for a dimensionality of 10, contd.

No. of	**f13**			**f14**		
Swrms	**S**	**Mean**	**Std.**	**S**	**Mean**	**Std.**
2	2000	5.577e-01	1.515e-01	10	2.664e+00	4.964e-01
3	25	5.635e-01	1.738e-01	**250**	**2.614e+00**	**4.659e-01**
5	**25**	**5.506e-01**	**1.567e-01**	1000	2.688e+00	2.619e-01
10	25	6.214e-01	2.097e-01	50	2.660e+00	2.949e-01



(a) Different balancing techniques

(b) Where each function is better optimized

Figure 3.11: Balancing techniques.

balancing techniques represented by points G, A, R, and S. The balancing graph could be used to establish some guidelines that might help in building a successful cooperative model.

The overall results show that half of the functions under study are best solved using 5 swarms. Another observation is that shifting or shifting-and-rotating a function would require increasing the number of swarms used in the model (Rastrigin and (F9, F10) or Griewank and (F7, F13)). This plot could be very useful when trying to optimize a new function. If one can find the closest one of the functions under study in this work to the new function depending on its underlying topology, the appropriate settings of the cooperative model to better optimize this function could be determined.

It is difficult to answer the question raised in the beginning of the section. There are other factors involved, which are the synchronization period and the topology of the optimized function.

3.3.7 Information Exchange

In this section, the model used has two cooperating swarms adopting the *lbest* model (local best) [53], where the particles are connected through a ring. This model suffers from a slow flow of information problem [54]. If one particle has useful information, it might take a while for other particles to benefit from it. That is why it is interesting to see if the cooperative approach is still efficient using this model.

The approach is compared to having a single swarm adopting the *lbest* model. The two cooperating swarms have 15 particles each. The neighborhood sizes selected are 2 for the Rastrigin and the Schwefel functions and 4 for the Ackley function (since these sizes produced the best results when having one swarm with 15 particles). The results are compared to a single swarm having 30 particles. Results of the single swarm approach are shown in Table 3.21.

Table 3.21: Results of the *lbest* model for a dimensionality of 10.

No. of Particles	**No. of Neighbors**	**Ackley**		**Rastrigin**		**Schwefel**	
		Mean	**Std.**	**Mean**	**Std.**	**Mean**	**Std.**
15	2	1.154e+00	7.592e-01	**5.974e+00**	**2.433e+00**	**6.590e+02**	**2.799e+02**
15	4	**9.086e-01**	**7.735e-01**	6.375e+00	2.718e+00	8.428e+02	4.344e+02
15	6	1.133e+00	9.519e-01	6.762e+00	3.696e+00	9.537e+02	4.368e+00
30	2	7.283e-01	6.684e-01	**4.081e+00**	**2.040e+00**	**6.238e+02**	**2.911e+02**
30	4	3.993e-01	6.663e-01	4.274e+00	1.642e+00	6.699e+02	3.601e+02
30	6	**2.931e-01**	**5.450e-01**	5.294e+00	1.949e+00	7.463e+02	3.719e+02

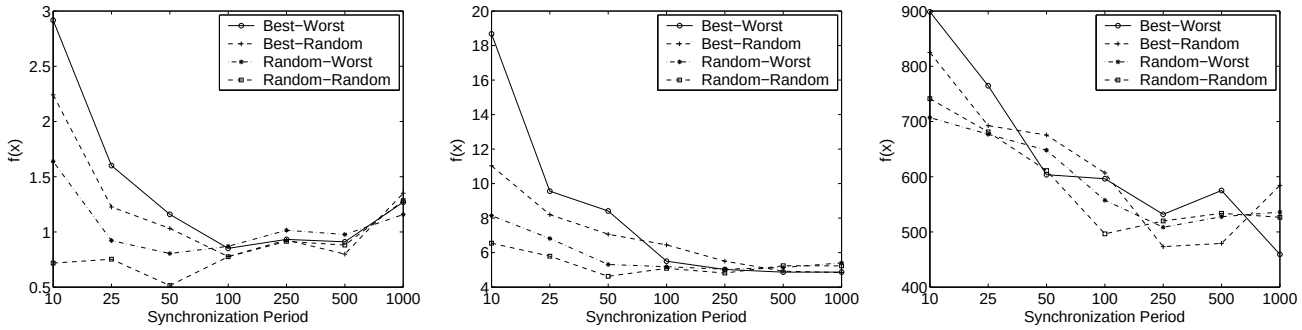
The best results obtained by applying different exchange approaches are shown in Table 3.22. The four different exchange approaches previously identified are applied by exchanging one particle ($p = 1$) both ways.

Table 3.22: Results of different exchange approaches for a dimensionality of 10.

Exchange Approach	**Ackley**			**Rastrigin**			**Schwefel**		
	S	**Mean**	**Std.**	**S**	**Mean**	**Std.**	**S**	**Mean**	**Std.**
gbest	50	1.621e+00	7.566e-01	250	8.632e+00	2.845e+00	250	5.082e+02	2.025e+02
B-W	100	8.509e-01	8.205e-01	500	4.871e+00	1.927e+00	1000	**4.594e+02**	**2.601e+02**
B-R	100	7.768e-01	7.410e-01	1000	4.842e+00	1.821e+00	250	4.733e+02	2.808e+02
R-W	50	8.046e-01	7.827e-01	250	5.038e+00	1.946e+00	250	5.084e+02	1.749e+02
R-R	50	**5.148e-01**	**7.558e-01**	50	**4.626e+00**	**1.820e+00**	100	4.965e+02	2.828e+02

The results show that exchanging any particle information both ways is better than just sharing the global best. It is also shown that all the different approaches could give comparable results at different synchronization periods, with the Best-Worst approach giving the best results at long synchronization periods and the Random-Random approach giving the best results at short synchronization periods. The Rastrigin function still gives the best results at longer synchronization periods than those needed for the Ackley function.

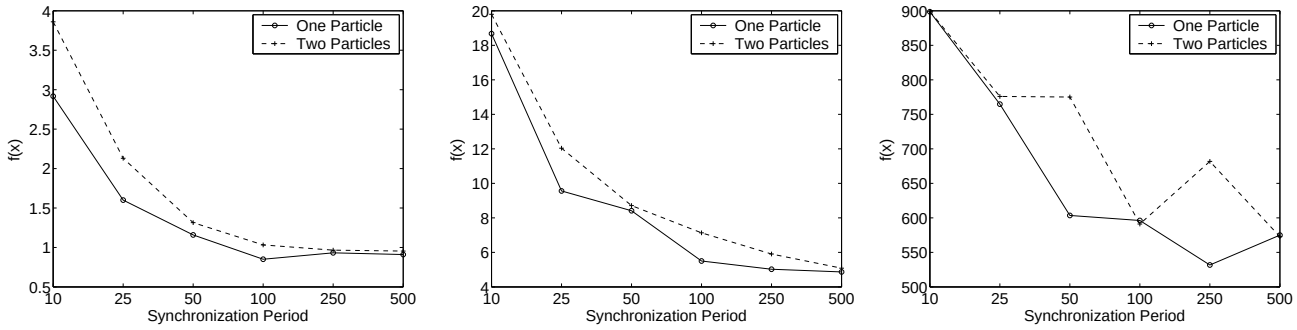
Figure 3.12 shows the comparison of the four information exchange approaches. The figure shows a similarity between the behaviors of the Ackley and the Rastrigin functions while the Schwefel function has a different behavior.



(a) Ackley (b) Rastrigin (c) Schwefel

Figure 3.12: Comparing four information exchange approaches.

Despite having the results improved when adopting the both ways sharing strategy, the results obtained when using a single swarm are still better for the Ackley and the Rastrigin functions. One approach that might improve the cooperative model performance is to exchange more particles ($p > 1$). The results are illustrated in Figure 3.13.



(a) Ackley (b) Rastrigin (c) Schwefel

Figure 3.13: Increasing the number of exchanged particles.

Unfortunately, the results show that increasing the number of particles still did not improve

the cooperative model performance. Exchanging more particles both ways will cause both swarms to have more similar particles, which seems not to be beneficial. New information exchange mechanisms are surely needed, as in [17], to overcome this drawback.

3.3.8 Increasing the Dimensionality

In this section, the experiments are repeated for a problem size of 30 performing 150000 function evaluations. The single swarm results are shown in Table 3.23. Table 3.24 shows the results of the cooperative model with 25 particles per swarm, which are plotted in Figure 3.14. The results show that the same behavior is observed at a higher dimensionality, increasing the synchronization period improves the performance up to a certain limit.

Table 3.23: Results of a single swarm for a dimensionality of 30.

Number of Particles	**Ackley**		**Rastrigin**		**Schwefel**	
	Mean	**Std.**	**Mean**	**Std.**	**Mean**	**Std.**
25	5.220e+00	7.920e-1	7.608e+01	1.449e+01	4.913e+03	1.356e+03
50	5.318e+00	8.746e-1	5.648e+01	1.959e+01	**4.068e+03**	**1.145e+03**
75	**4.733e+00**	**1.128e+00**	**5.112e+01**	**1.502e+01**	4.190e+03	8.229e+02

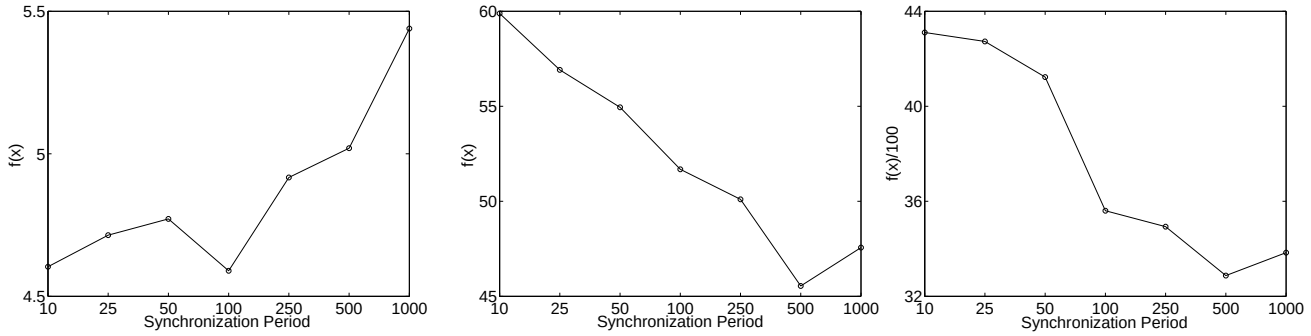
Table 3.24: Results of the cooperative model for a dimensionality of 30.

Benchmark Function	**Synchronization Period**	**Mean**	**Std.**	**Compared to Single Swarm**
Ackley	100	4.590e+00	7.868e-01	Similar to 75 particles
Rastrigin	500	4.555e+01	1.059e+01	Similar to 75 particles
Schwefel	500	3.287e+03	7.182e+02	Better

Since the Rastrigin function is harder than the Ackley function, the cooperative model needed a longer synchronization period, in both dimensionalities, than what is needed for the Ackley function in order to produce the best results. This emphasizes that for harder functions more separation is needed.

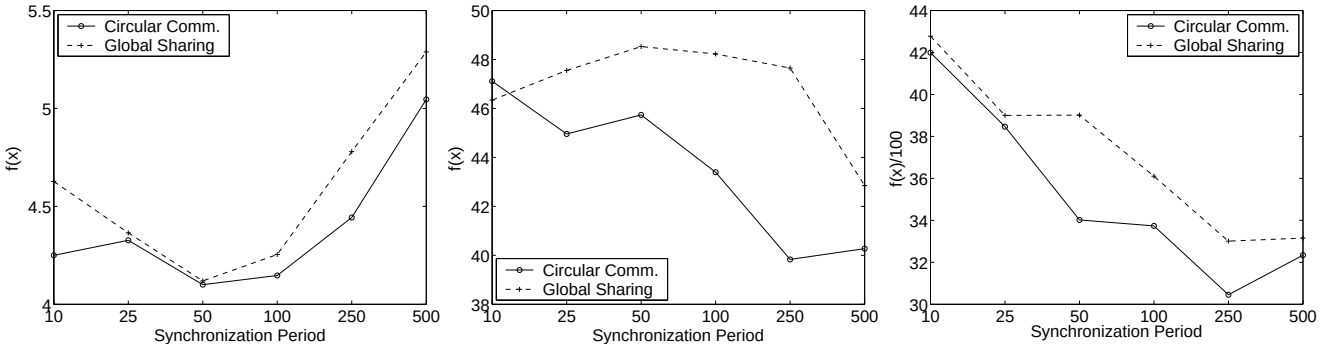
As for the communication topology, the results for the higher dimensionality are plotted in Figure 3.15. The results show the same behavior where the circular communication approach outperforming the global sharing mechanism.

To study the number of swarms factor, the experiments are run using 2, 3, 5, and 10 cooperating swarms adopting the circular communication approach, while keeping the number of particles fixed



(a) Ackley (b) Rastrigin (c) Schwefel

Figure 3.14: Synchronization period effect for a dimensionality of 30.



(a) Ackley (b) Rastrigin (c) Schwefel

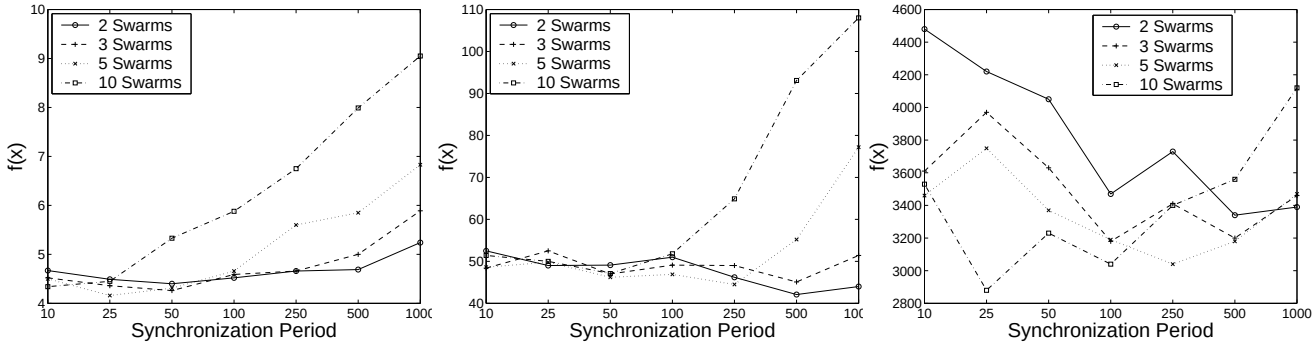
Figure 3.15: Comparing two neighborhood topologies for a dimensionality of 30.

at 60. The results shown in Table 3.25 show that the relation between the number of swarms and the synchronization period still holds for the higher dimensionality. The results show that increasing the number of swarms limits the synchronization period needed for achieving the best result. Again, the Rastrigin function is better solved at longer synchronization periods than the Ackley function in all the experiments. For the Schwefel function, the best result is still obtained by increasing the number of swarms up to 10.

Figure 3.16 illustrates the overall behavior of the cooperative model for the higher dimensionality. The same behavior from the previous experiments still holds as the model has a similar behavior when applied to the Ackley and the Rastrigin functions. For these functions, the results drastically deteriorate if both the number of swarms and the synchronization period are increased. As for the the Schwefel function, the overall behavior looks different as shown before.

Table 3.25: Varying the number of swarms for the multimodal functions for a dimensionality of 30.

No. of Swarms	Ackley			Rastrigin			Schwefel		
	S	**Mean**	**Std.**	**S**	**Mean**	**Std.**	**S**	**Mean**	**Std.**
2	50	4.399e+00	6.483e-01	**500**	**4.208e+01**	**13.772**	500	3.337e+03	3.957e+02
3	50	4.256e+00	7.983e-01	500	4.509e+01	9.437	500	3.202e+03	6.926e+02
5	**25**	**4.162e+00**	**6.351e-01**	250	4.450e+01	10.419	250	3.041e+03	5.965e+02
10	10	4.344e+00	5.436e-01	50	4.712e+01	10.722	**25**	**2.879e+03**	**5.329e+02**



(a) Ackley (b) Rastrigin (c) Schwefel

Figure 3.16: Results of increasing the number of swarms for a dimensionality of 30.

3.4 Conclusion

This chapter starts by giving a definition of what is meant by a cooperative PSO algorithm based on the models surveyed in the previous chapter. The given definition helps in identifying key design decisions (parameters settings) that affect the performance of any cooperative PSO algorithm. These decisions mainly give answers to four important questions; Which information to share? When to share it? Whom to share it with? and What to do with it? The chapter summarizes the design decisions taken by the different surveyed models.

A complete empirical study is then carried on one cooperative PSO model to observe how the performance is influenced by these parameters. The parameters experimented with are the synchronization period, the neighborhood topology, the number of cooperating swarms, the type of exchanged information, and the taken action. The experiments show that the suitable tuning of these factors can improve the performance of the cooperative model.

The experiments show that increasing the synchronization period up to a certain limit, provides better results in solving multimodal functions. A different number of conclusions were reached as

it was shown that adopting a 2-swarm cooperative model with n particles each is always better than having a single swarm with $2n$ particles. Also when adopting a small population 2-swarm cooperative model, the results obtained could be easily as good as (if not better than) the best results achieved by a single swarm with a huge population.

When having more than two cooperating swarms, the circular communication strategy provided better results than global sharing. This strategy increases the probability of the swarms following different directions in the search and is more suitable for optimizing multimodal functions.

If the number of particles is kept fixed, increasing the number of cooperating swarms while having a long synchronization period is not a smart choice. The experiments show that increasing one factor should put a limit on the other as when the number of swarms increases, the best solution is usually obtained at shorter synchronization periods. Although different functions are better solved using different settings of the number of swarms and the synchronization period, it is shown that a reasonable choice when attempting to optimize a new function is to use five swarms, as this value produced the best results for most of the functions under study. Another option is to find out the suitable settings for optimizing this function by finding the closest one to it from the functions under study and use the same settings.

An important behavior that was observed through all the conducted experiments is that more separation is needed between the cooperating swarms (longer synchronization periods) as the function being optimized gets harder (from the topology point of view).

For the functions with similar topologies, the Ackley and the Rastrigin functions, the cooperative model under study had similar behaviors against changing the identified parameters. The Rastrigin function is always better solved in all experiments at longer synchronization periods than those needed for the Ackley function.

Chapter 4

Particle Swarm Optimization Based on Probabilistic Models

The motivation of the work introduced in this chapter is mainly to investigate the idea of exchanging probabilistic models information between the cooperating swarms instead of the exchange of *gbest* information or the exchange of a group of particles that were studied in the previous chapter. In order to achieve this, the chapter gives a brief introduction to Estimation of Distribution Algorithms (EDAs) and the different methods previously adopted to combine PSO and EDAs. A new combination approach is also proposed, which borrows ideas from Population-Based Incremental Learning (PBIL). The new method is implemented and compared against the other PSO and EDA hybrids. All these models serve as the basis of the cooperative PSO approach that adopts the migration of probabilistic models introduced in the next chapter.

4.1 Estimation of Distribution Algorithms

Estimation of distribution algorithms (EDAs) [55], are evolutionary algorithms that try to estimate the probability distribution of the good individuals in the population. EDAs try to estimate this probability distribution by using selected individuals, from the current population, to construct a probabilistic model. This model is consequently used to generate a new population replacing the current one and so on. Hence, EDAs maintain a continuously updated probabilistic model from one generation to the next. Although, it has been originally introduced to tackle combinatorial optimization problems, a recent numerical application has been proposed as well [56–59].The general steps for an EDA is shown in Algorithm 4.1.

Algorithm 4.1 Estimation of Distribution Algorithm (EDA).

1: $P \Leftarrow$ Initialize the population
2: Evaluate the initial population
3: **while** $iter\_number \leq Max\_iterations$ **do**
4: $P_s \Leftarrow$ Select the top s individuals
5: $M \Leftarrow$ Estimate a new Model from P_s
6: $P_n \Leftarrow$ Sample n individuals from M
7: Evaluate P_n
8: $P \Leftarrow$ Select n individuals from PUP_n
9: iter_number = iter_number + 1
10: **end while**

4.2 PSO Based on Probabilistic Models

This section surveys the two previous attempts to introduce the concepts of EDAs into PSO in order to improve its performance.

4.2.1 EDPSO

An estimation of distribution particle swarm optimizer (EDPSO) was proposed by Iqbal and Montes de Oca [60]. The method borrowed some ideas from a development in ACO for solving continuous optimization problems [61]. The approach relies on estimating the joint probability distribution for one dimension at a time using mixtures of weighted Gaussian functions. The Gaussian functions are defined through an archive of k solutions ($pbests$ of the particles). For each dimension d, the dimension is either updated using PSO equations or by sampling a Gaussian distribution selected from the archive. The values of this dimension d across all the solutions in the archive compose the vector $\boldsymbol{\mu_d}$, which is the vector of means for the univariate Gaussian distributions:

$$\boldsymbol{\mu_d} =< pbest_{1d}, pbest_{2d}, ..., pbest_{kd} > \tag{4.1}$$

To select one of these distributions, the weights vector $\mathbf{w}$, which holds the weights associated with each distribution, is calculated. This is done by sorting the solutions according to their fitness, with the best solution having a rank of 1. A weight is calculated for each solution as follows:

$$\begin{aligned} \mathbf{w} &=< w_1, w_2, ..., w_k > \\ w_l &= \frac{1}{qk\sqrt{2\pi}} e^{-\frac{(l-1)^2}{2q^2k^2}} \end{aligned} \tag{4.2}$$

where q determines how much we prefer good solutions and l is the solution rank.

The Gaussian function to be used is selected probabilistically. The probability of selecting a certain Gaussian function is proportional to its weight. This probability is calculated as follows:

$$\begin{aligned} \mathbf{p} &= < p_1, p_2, ..., p_k > \\ p_l &= \frac{w_l}{\sum_{r=1}^{k} w_r} \end{aligned} \tag{4.3}$$

After selecting a certain Gaussian function G_d denoted by its mean $pbest_{gd}$, where $1 < g < k$, the standard deviation for this functions is calculated as:

$$\sigma_{gd} = \xi \sum_{i=1}^{k} \frac{|pbest_{id} - pbest_{gd}|}{k-1} \tag{4.4}$$

where ξ is a parameter to balance the exploration-exploitation behaviors.

Finally the selected Gaussian function is evaluated (not sampled) to generate a value r in order to probabilistically move the particle. This is done by generating a uniformly distributed random number U(0,1). If it is less than r, the particle moves using the normal PSO equations. Otherwise, the Gaussian function is sampled to move the particle. The steps are shown in Algorithm 4.2.

4.2.2 EDA-PSO

A hybrid EDA-PSO approach was proposed in [62] and shown in Algorithm 4.3. The algorithm works by sampling an independent univariate Gaussian distribution based on the best half of the swarm. The mean and standard deviation of the model is calculated in every iteration as:

$$\begin{aligned} \boldsymbol{\mu} &= \frac{1}{M} \sum_{i=1}^{M} \mathbf{x}_i \\ \sigma_j &= \sqrt{\frac{1}{M} \sum_{i=1}^{M} (\mathbf{x}_{ij} - \mu_j)^2}, \end{aligned} \tag{4.5}$$

where $M = N/2$ for a swarm with N particles and i is the particle number.

The choice of whether to update the particle using the normal PSO equations or to sample the particle using the estimated distribution is made with a probability p, referred to as the *participation ratio*. If $p = 0$, the algorithm will behave as a pure EDA algorithm and if $p = 1$, it

Algorithm 4.2 The EDPSO algorithm.

Require: Max_Function_Evaluations

Initialize the swarm
Max_Iterations = $\frac{Max\_Function\_Evaluations}{Num\_Particles}$
iter_number = 1
while $iter\_number \leq Max\_Iterations$ **do**
  Update the swarm
  Rank the particle's using pbests information
  Compute weights vector **w**
  Compute probabilities vector **p**
  for every particle i **do**
    for each dimension d **do**
      Update v_{id} and x_{id}
      Select a Gaussian function according to p_i
      Calculate σ_{gd}
      Prob_move = $\sigma_{gd}\sqrt{2\pi}G_d(x_{id})$
      if $U(0,1) < Prob\_move$ **then**
        continue
      else
        x_{id} = Gauss(s_{gd},σ_{gd})
      end if
    end for
  end for
  iter_number = iter_number + 1
end while
return *gbest*

will be a pure PSO algorithm. In the hybrid approach, where $0 < p < 1$, each particle is either totally updated by the PSO equations or totally sampled from the estimated distribution (not on a dimension-by-dimension basis as in EDPSO). Finally, the particle gets updated only if its fitness improves. The authors also proposed different approaches in order to adaptively set the parameter p. These approaches depend on the success rate of both the PSO and EDA parts in improving a particles fitness:

- The first approach is the *Generation based*, where the success rates are calculated based on

the information gathered during the last generation,

- The second approach is the *All historical information*, where the success rates are calculated based on the information gathered during the entire search,
- The final approach is the *Sliding window*, where the success rates are calculated considering only the information in the last m generations.

Algorithm 4.3 The EDA-PSO algorithm.

Require: Max_Function_Evaluations

```
Initialize the swarm
Max_Iterations = Max_Function_Evaluations / Num_Particles
iter_number = 1
while iter_number ≤ Max_Iterations do
    Calculate μ and σ using top N/2 particles
    for every particle i do
        if U(0,1) < p then
            candidate_particle = PSO equations
        else
            candidate_particle = Gauss(μ,σ)
        end if
        if candidate_particle has a better fitness then
            particle i = candidate_particle
        end if
    end for
    iter_number = iter_number + 1
end while
return gbest
```

4.3 PSO with Varying Bounds

A population-based incremental learning (PBIL) approach for continuous search spaces was proposed in [57]. The algorithm explored the search space by dividing the domain of each gene into two equal intervals referred to as the *low* and *high* intervals. A probability h_d, which is initially

set to 0.5, is the probability of gene number d being in the *high* interval as shown:

$$x_d \in [a, b], h_d = Probability(x_d > \frac{a+b}{2}) \tag{4.6}$$

After each generation, this distribution is updated according to the gene values of the best individual using the following formula:

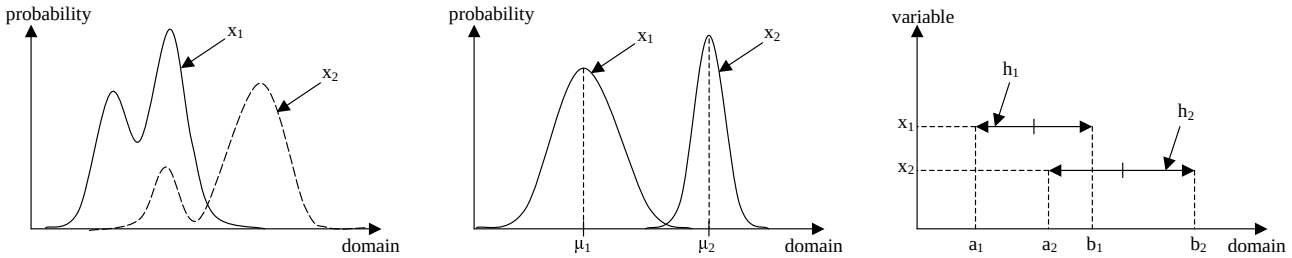
$$p = \begin{cases} 0 & \texttt{if} \quad x_d^{max} < \frac{a+b}{2} \\ 1 & otherwise \end{cases} \tag{4.7}$$
$$h_d^{t+1} = (1 - \alpha) * h_d^t + \alpha * p$$

where α is the *relaxation factor* and t is the iteration number. If h_d gets below h_{dmin} or above h_{dmax}, the population gets re-sampled in the corresponding interval, $[a, \frac{a+b}{2}]$ or $[\frac{a+b}{2}, b]$, respectively.

In this work, the concepts of PBIL are introduced into PSO. At the beginning, the particles are initialized in the predefined domain. After every iteration, the probability h_d of each dimension d gets adjusted according to the probability of this dimension value being in the *high* interval of the defined domain. This probability is calculated using information from all the particles and not only *gbest* to prevent premature convergence. When h_d gets specific enough, the domain of dimension d is adjusted accordingly and h_d gets re-initialized to 0.5. In this model, different dimensions might end up having different domains and different velocity bounds which does not happen in normal PSO.

The steps taken by PSO_Bounds is shown in Algorithm 4.4 where x_{dmin} and x_{dmax} refer to the minimum and maximum bounds for dimension d while v_{dmin} and v_{dmax} refer to the velocity bounds.

Figure 4.1 [63] illustrates the approaches taken by the different algorithms to model the distribution of good solutions across every dimension.



(a) EDPSO (b) EDA-PSO (c) PSO_Bounds

Figure 4.1: Probabilistic models.

Algorithm 4.4 The PSO_Bounds algorithm.

Require: Max_Function_Evaluations

Initialize the swarm
Max_Iterations = $\frac{Max\_Function\_Evaluations}{Num\_Particles}$
iter_number = 1
while $iter\_number \leq Max\_Iterations$ **do**
    Update the swarm
    for each dimension d **do**
        Calculate the probability of dimension d
        update h_d
        if $h_d < h_{dmin}$ **then**
            $x_{dmax} = b = \frac{a+b}{2}$
            Update v_{dmin} and v_{dmax}
            $h_d = 0.5$
        else if $h_d > h_{dmax}$ **then**
            $x_{dmin} = a = \frac{a+b}{2}$
            Update v_{dmin} and v_{dmax}
            $h_d = 0.5$
        end if
    end for
    iter_number = iter_number + 1
end while
return *gbest*

4.4 Results and Discussions

4.4.1 Experimental Settings

Table 4.1 shows the parameter settings used for applying the algorithms under study. For all experiments, all the particles have been randomly initialized in the specified domain using uniform distribution. The values for q and ξ are the same as was proposed in [60] and the value for p is set adaptively using the $allhistoricalinformation$ approach, as it was found to be the best one based on our experiments. The values for (α, h_{dmin}, h_{dmax}) are changed from (0.01, 0.1, 0.9) in [57] to (0.1, 0.2, 0.8) to allow a faster process of varying the bounds. The experiments are conducted for a problem dimensionality of 10, 30, and 50 with 40 particles in the swarm performing 100000,

100000, and 200000 function evaluations, respectively. The results reported are the averages taken over 30 runs.

Table 4.1: Parameter settings.

Model	**Parameter**	**Value**
Normal PSO	w	0.9 to 0.1
	c1 and c2	2
EDPSO	q	0.1
	ξ	0.85
EDA-PSO	p	Adaptive - all historical information
PSO_Bounds	α	0.1
	h_{dmin}	0.2
	h_{dmax}	0.8

The experiments are conducted using both the classical benchmark functions and the CEC05 benchmark functions f6-f14. In order to constrain the particles movement within the specified domain for the CEC05 functions, any violating particle gets its position randomly re-initialized inside the specified domain. The error values $f(x) - f(x*)$ are reported, where $x*$ is the global optimum.

In [62], the values for $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ are calculated using the best half of the swarm. The authors in [64] proposed calculating $\boldsymbol{\sigma}$ using the whole population instead, which is found to produce better results due to the induced diversity avoiding premature convergence. The same approach is used in this work when applying the EDA-PSO algorithm.

4.4.2 Experimental Results

Tables 4.2 and 4.3 show the results obtained by applying EDPSO, EDA-PSO and PSO_Bounds to the classical and CEC05 functions for different problem sizes. The PSO_Bounds algorithm is not applied for f7 as this function is not bounded by a specified domain (the bounds shown in Table 3.8 are only used as an initialization range).

As shown in Tables 4.2 for the classical functions, both EDPSO and EDA-PSO outperform PSO_Bounds. The reason for this is that the global optimum is at the center of the search space and the Gaussian model adopted by these algorithms along with the uniform distribution used in initializing the particles make it very easy for these algorithms to reach better results.

Table 4.2: Results of all the algorithms for the classical functions.

Benchmark Function	Dim.	EDPSO		EDA-PSO		PSO_Bounds	
		Mean	**Std.**	**Mean**	**Std.**	**Mean**	**Std.**
Spherical	10	**9.881e-324**	**0**	8.400e-266	0	5.087e-03	2.786e-02
Rosenbrock		**5.519e-06**	**1.044e-05**	7.827e-02	8.422e-02	7.744e-01	5.857e-01
Griewank		2.084e-02	1.447e-02	**7.882e-03**	**7.325e-03**	1.229e-01	5.988e-02
Ackley		**5.887e-16**	**2.006e-31**	1.268e+00	2.258e-15	8.606e-02	4.708e-01
Rastrigin		**3.051e+00**	**1.609e+00**	4.013e+00	1.998e+00	7.131e+00	2.172e+00
Spherical	30	3.698e-67	2.026e-66	**4.234e-141**	**1.425e-140**	5.416e+02	3.674e+02
Rosenbrock		**9.562e-01**	**2.042e-01**	**1.123e+00**	**4.552e-01**	1.707e+01	4.633e+00
Griewank		1.479e-03	3.462e-03	**0**	**0**	4.871e+00	2.021e+00
Ackley		**4.378e-015**	**9.014e-016**	1.586e+00	9.034e-16	5.467e+00	1.137e+00
Rastrigin		**1.791e+01**	**4.222e+00**	3.4067e+01	2.922e+01	6.799e+01	1.339e+01
Spherical	50	1.104e-59	3.644e-59	**2.811e-103**	**1.539e-102**	2.979e+03	1.131e+03
Rosenbrock		**2.078e+00**	**3.954e-01**	**1.565e+00**	**2.745e+00**	3.131e+01	7.791e+00
Griewank		**3.286e-04**	**1.800e-03**	2.132e-03	6.314e-03	2.697e+01	8.899e+00
Ackley		**7.694e-15**	**1.319e-15**	1.641e+00	2.258e-16	9.068e+00	9.036e-01
Rastrigin		**4.016e+01**	**8.593e+00**	**4.630e+01**	**1.410e+01**	1.457e+02	1.891e+01

For the CEC05 functions shown in Table 4.3, the results show that PSO_Bounds has much better performance compared to the other algorithms across the different problem size.

Table 4.4 summarizes the comparison between all the algorithms based on the results shown in Tables 4.2 and 4.3. The upper bound for the number of cases is 15 (5 functions in 3 problem sizes) in the classical functions and 21 (7 functions in 3 problem sizes) in the CEC05 functions.

The convergence behavior shown in Figure 4.2 and Figure 4.3 illustrates that PSO_Bounds usually has a slow speed of convergence compared with the other algorithms. It only has the fastest speed of convergence in both f6 and f9 where it does not produce good results. Convergence figures also show that both EDPSO and EDA-PSO have a very similar behavior on most of the functions.

In [65], the authors stated that "modern research performed using only swarms with a global topology is incomplete at best". For this reason, the experiments are rerun again for all the algorithms using the *lbest* (local best) model [53]. Table 4.5 and Table 4.6 show the obtained results. The results show that all the algorithms exhibit the same performance compared to each other as in the case of using the *gbest* model.

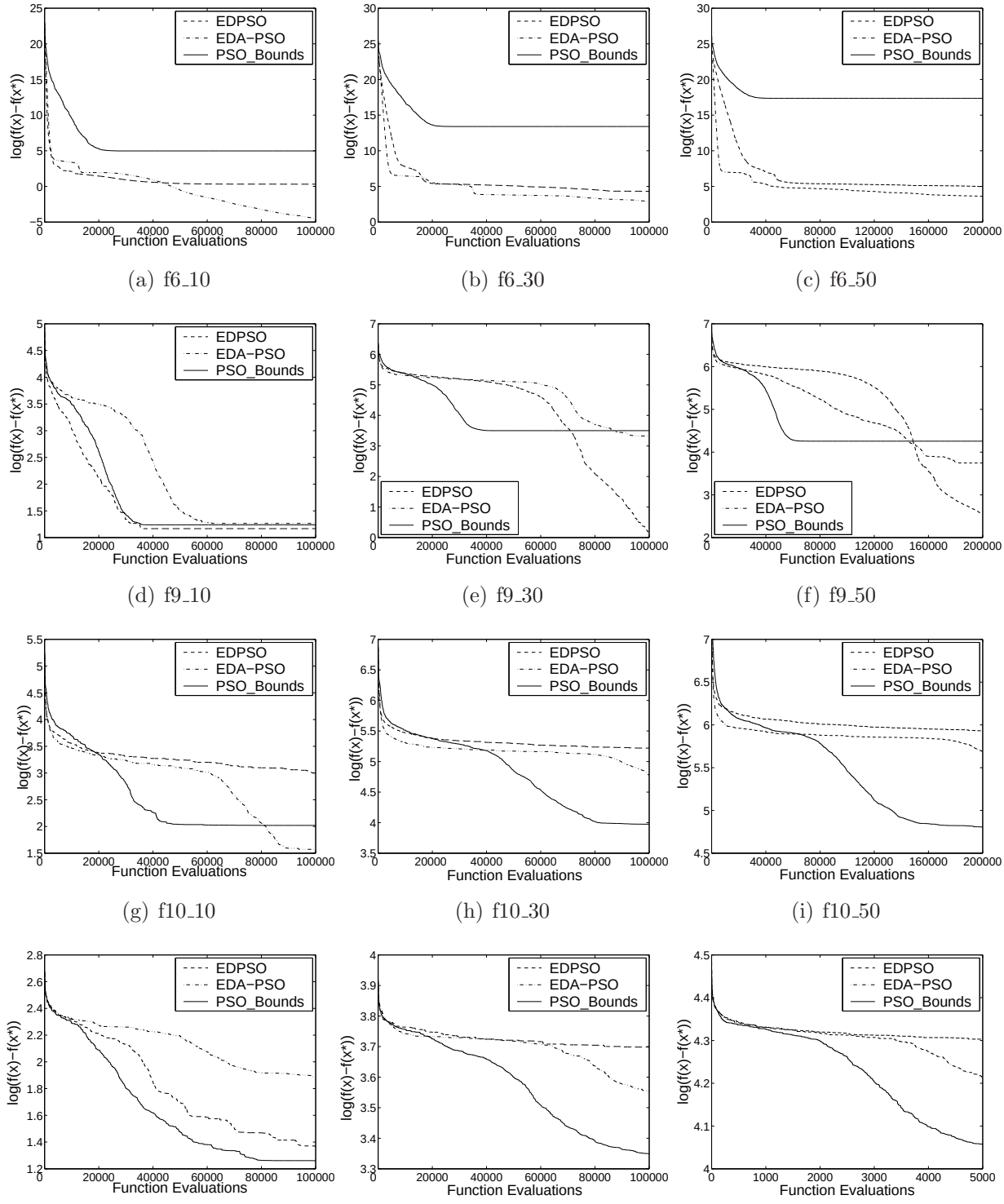
Table 4.7 summarizes the comparison between all the algorithms based on the results shown

Table 4.3: Results of all the algorithms for the CEC05 functions.

Benchmark Function	Dim.	EDPSO		EDA-PSO		PSO_Bounds	
		Mean.	Std.	Mean	Std.	Mean	Std.
f6	10	1.375e+00	4.557e+00	**1.123e-02**	**1.626e-02**	1.451e+02	2.218e+02
f7		2.687e-01	2.258e-01	**1.927e-01**	**1.905e-01**	-	-
f9		**3.217e+00**	**1.604e+00**	4.046e+00	2.277e+00	**3.454e+00**	**1.471e+00**
f10		1.989e+01	6.327e+00	**4.819e+00**	**3.642e+00**	7.543e+00	4.528e+00
f11		3.868e+00	3.859e+00	6.588e+00	1.340e+00	**3.529e+00**	**1.730e+00**
f12		2.919e+04	7.054e+03	1.616e+04	6.334e+03	**4.243e+03**	**5.001e+03**
f13		1.194e+00	5.372e-01	8.465e-01	3.968e-01	**6.904e-01**	**1.770e-01**
f14		2.429e+00	5.255e-01	2.667e+00	5.991e-01	**2.365e+00**	**5.792e-01**
f6	30	7.522e+01	1.007e+02	**1.716e+01**	**2.011e+01**	6.602e+05	1.841e+06
f7		**8.700e-03**	**5.920e-03**	1.300e-02	7.589e-03	-	-
f9		**1.175e+00**	**2.044e+00**	2.789e+01	6.498e+00	3.315e+01	7.072e+00
f10		1.850e+02	1.348e+01	1.187e+02	6.191e+01	**5.556e+01**	**2.068e+01**
f11		4.028e+01	1.676e+00	3.494e+01	2.674e+00	**2.849e+01**	**3.897e+00**
f12		1.129e+06	1.266e+05	9.219e+05	2.060e+05	**2.941e+05**	**2.155e+05**
f13		1.489e+01	1.497e+00	7.942e+00	4.688e+00	**4.333e+00**	**7.852e-01**
f14		1.334e+01	2.309e-01	1.325e+01	2.933e-01	**1.245e+01**	**6.541e-01**
f6	50	1.429e+02	2.023e+02	**3.725e+01**	**4.515e+01**	3.458e+07	4.913e+07
f7		**3.000e-03**	**5.813e-03**	9.867e-03	1.374e-02	-	-
f9		**1.282e+01**	**6.519e+00**	4.232e+01	1.080e+01	7.047e+01	1.338e+01
f10		3.765e+02	1.520e+01	2.931e+02	8.820e+01	**1.222e+02**	**2.553e+01**
f11		7.393e+01	1.266e+00	6.744e+01	3.031e+00	**5.778e+02**	**6.800e+00**
f12		5.760e+06	3.738e+05	3.965e+06	1.259e+06	**1.254e+05**	**1.167e+05**
f13		3.057e+01	2.701e+00	1.696e+01	1.109e+01	**9.327e+00**	**2.030e+00**
f14		2.310e+01	2.551e-01	2.282e+01	3.451e-01	**2.237e+01**	**4.455e-01**

Table 4.4: Comparison between all the algorithms using the *gbest* model.

Algorithm	Classical Functions		CEC05 Functions		Total Number of Cases
	No. of Cases	Best in	No. of Cases	Best in	
PSO_Bounds	-	-	15	f11, f12 f13, f14	15
EDA-PSO	7	-	5	f6	12
EDPSO	11	Rosenbrock Ackley, Rastrigin	5	-	16



(a) f6_10 (b) f6_30 (c) f6_50

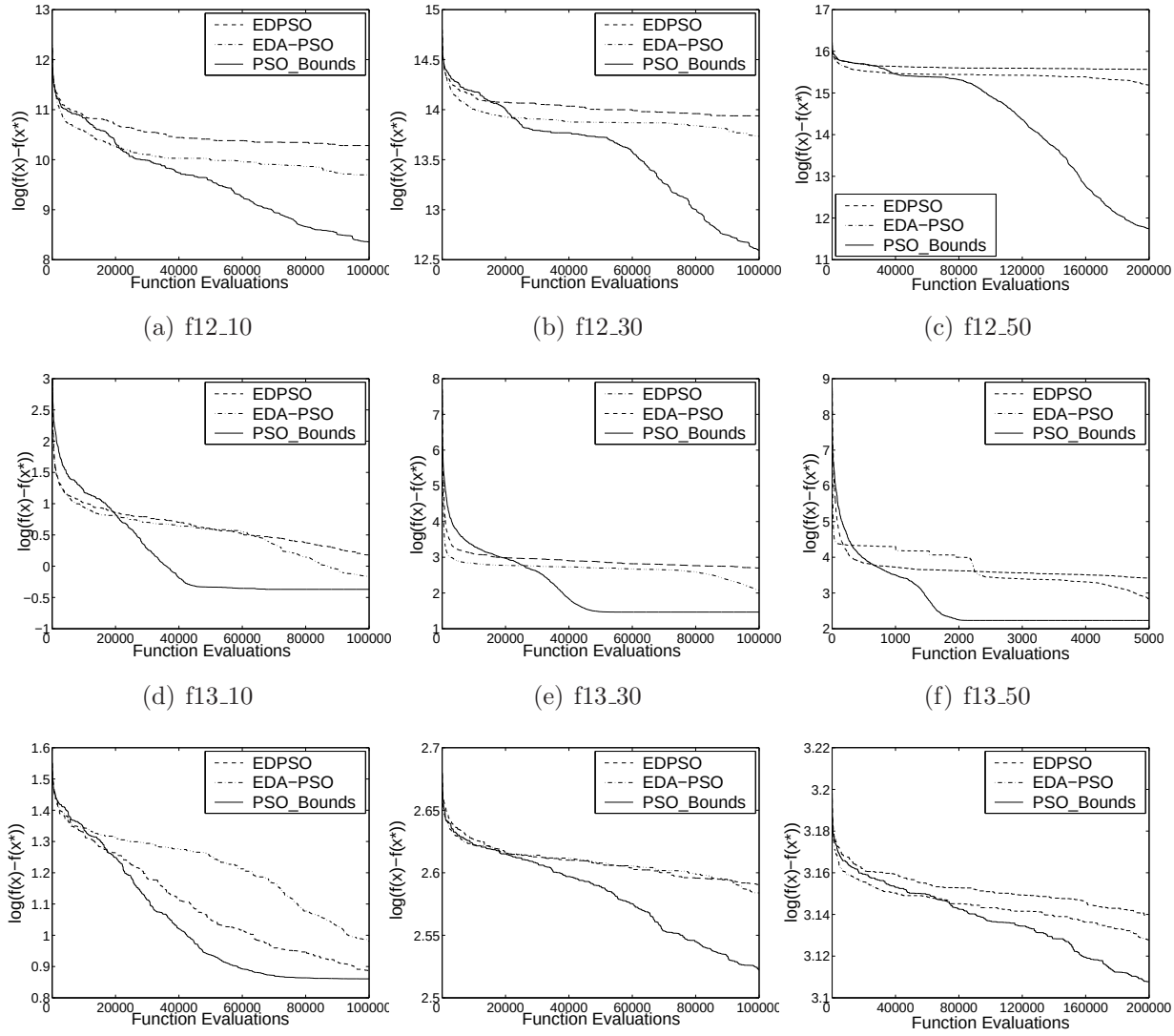
(d) f9_10 (e) f9_30 (f) f9_50

(g) f10_10 (h) f10_30 (i) f10_50

(j) f11_10 (k) f11_30 (l) f11_50

Figure 4.2: Convergence behavior of all the algorithms for the CEC05 functions.

in Tables 4.5 and 4.6. The results emphasize that the performance of these algorithms (compared to each other) is the same regardless of the underlying population topology.



(a) f12_10 (b) f12_30 (c) f12_50

(d) f13_10 (e) f13_30 (f) f13_50

(g) f14_10 (h) f14_30 (i) f14_50

Figure 4.3: Convergence behavior of all the algorithms for the CEC05 functions, contd.

4.5 Conclusion

This chapter gives a brief introduction to Estimation of Distribution Algorithms (EDAs) and surveys the different methods previously adopted to combine PSO and EDAs. The chapter proposes a PSO algorithm that borrows ideas from PBIL. The proposed algorithm is compared to other existing PSO and EDA hybrids on a number of well-known benchmark optimization functions. The proposed algorithm is shown to be better than existing models in the difficult multimodal optimization functions across the different problem sizes. The experiments also show that the proposed algorithm has a generally slower speed of convergence than the other existing approaches

Table 4.5: Results of all the algorithms using the *lbest* model for the classical functions.

Benchmark Function	**Dim.**	**EDPSO_L**		**EDA-PSO_L**		**PSO_Bounds_L**	
		Mean	**Std.**	**Mean**	**Std.**	**Mean**	**Std.**
Spherical	10	**0**	**0**	**3.850e-267**	**0**	**3.194e-17**	**1.722e-16**
Rosenbrock		**4.019e-03**	**5.097e-03**	1.029e-01	5.276e-02	2.390e-01	1.781e-01
Griewank		1.682e-02	1.194e-02	**4.959e-03**	**7.767e-03**	4.881e-02	1.517e-02
Ackley		**5.887e-15**	**2.006e-31**	1.268e+00	2.258e-16	5.037e-11	9.260e-11
Rastrigin		**3.118e+00**	**1.5180e+00**	**3.263e+00**	**1.885e+00**	**3.798e+00**	**1.444e+00**
Spherical	30	**6.031e-94**	**3.205e-95**	**7.020e-141**	**9.280e-141**	5.183e-01	1.811
Rosenbrock		**1.076e+00**	**1.779e-01**	**1.742e+00**	**2.524e+00**	1.085e+01	3.416e+00
Griewank		**2.052e-03**	**4.970e-03**	3.288e-02	1.801e-03	7.657e-02	8.055e-02
Ackley		**4.141e-015**	**0**	1.586	3.651e-16	6.166e-01	7.217e-01
Rastrigin		**1.523e+01**	**3.999e+00**	4.472e+01	3.057e+01	4.192e+01	7.900e+00
Spherical	50	**3.055e-82**	**1.256e-81**	**3.700e-11**	**2.026-10**	14.233	33.711
Rosenbrock		**2.104e+00**	**2.490e-01**	6.237e+00	1.035e+01	2.356e+01	4.209e+00
Griewank		**1.232e-03**	**3.284e-03**	**2.919e-03**	**1.465e-02**	6.929e-01	3.780e-01
Ackley		**6.865e-15**	**1.528e-15**	1.641e+00	2.258e-16	2.200e+00	6.118e-01
Rastrigin		**3.270e+01**	**7.202e+00**	7.481e+01	5.061e+01	9.140e+01	1.469e+01

and that its performance (compared to the other existing approaches) is robust against changing the underlying population topology.

Table 4.6: Results of all the algorithms using the *lbest* model for the CEC05 functions.

Benchmark Function	Dim.	EDPSO_L		EDA-PSO_L		PSO_Bounds_L	
		Mean	Std.	Mean	Std.	Mean	Std.
f6	10	6.554e+00	1.936e+01	**2.092e-01**	**7.899e-01**	1.497e+01	25.785
f9		2.919e+00	1.566e+00	3.310e+00	1.259e+00	**8.025e-01**	**9.603e-01**
f10		1.946e+01	6.939e+00	1.105e+01	5.716e+00	**6.712e+00**	**3.003e+00**
f11		7.292e+00	3.473e+00	6.196e+00	8.1308e-1	**4.480e+00**	**1.027e+00**
f12		2.729e+04	7.468e+03	1.877e+04	6.712e+3	**6.535e+03**	**2.841e+03**
f13		1.435e00	4.549e-01	1.224e+00	3.724e-01	**6.422e-01**	**1.390e-01**
f14		**2.204e+00**	**5.145e-01**	**2.910e+00**	**2.684e-01**	**2.777e+00**	**3.261e-01**
f6	30	**8.592e+01**	**1.305e+02**	**7.063e+01**	**4.586e+01**	8.883e+03	3.632e+04
f9		**1.605e+01**	**5.372e+00**	4.208e+01	2.742e+01	2.536e+01	4.694e+00
f10		1.778e+02	9.953e+00	1.608e+02	1.719e+01	**1.384e+02**	**1.864e+01**
f11		4.043e+01	1.148e+00	3.641e+01	2.107e+00	**3.163e+01**	**2.479e+00**
f12		1.140e+06	1.148e+05	9.571e+05	1.696e+05	**4.978e+05**	**1.443e+05**
f13		1.447e+01	1.328e+00	1.156e+01	2.639e+00	**4.755e+00**	**9.558e-01**
f14		1.347e+01	1.873e-01	1.327e+01	2.282e-01	**1.302e+01**	**2.674e-01**
f6	50	**6.550e+01**	**5.446e+01**	**6.789e+01**	**4.228e+01**	1.967e+06	1.012e+07
f9		**3.250e+01**	**6.450e+01**	5.334e+01	2.326e+01	5.547e+01	9.813e+00
f10		3.629e+02	1.624e+01	3.359e+02	1.660e+01	**2.879e+02**	**4.048e+01**
f11		7.381e+01	1.911e+00	6.926e+01	2.552e+00	**6.184e+01**	**4.231e+00**
f12		5.631e+06	4.676e+05	4.725e+06	5.695e+05	**1.896e+06**	**3.675e+05**
f13		2.738e+01	4.029e+00	2.446e+01	4.217e+00	**1.062e+01**	**2.183e+00**
f14		**2.316e+01**	**1.693e-01**	**2.292e+01**	**2.184e-01**	**2.261e+01**	**2.103e-01**

Table 4.7: Comparison between all the algorithms using the *lbest* model.

Algorithm	Classical Functions		CEC05 Functions		Total Number of Cases
	No. of Cases	Best in	No. of Cases	Best in	
PSO_Bounds	2	-	16	f10, f11 f12, f13	18
EDA-PSO	7	Spherical	5	f6	12
EDPSO	14	Spherical, Rosenbrock Ackley, Rastrigin	6	-	20

Chapter 5

A Heterogeneous Cooperative Particle Swarm Optimizer with Migrated Probability Models

This chapter introduces a new cooperative PSO that is based on the exchange of probability models. The chapter starts with surveying the different parallel EDAs proposed in the literature relying on either exchanging individuals or exchanging probability models. The new heterogeneous cooperative PSO/EDA algorithm is then proposed. The model is considered heterogeneous as the cooperating PSO/EDA algorithms use different methods to sample the search space. The behavior of the proposed model is discussed based on the solutions obtained and the convergence behavior. A simple adaptive version to control the information flow between the cooperating swarms is also presented. Both the non-adaptive and the adaptive versions are implemented and compared to existing PSO cooperative approaches using a suite of well-known benchmark optimization functions.

5.1 Parallel EDAs

A recent research direction in the previous 4 years was to introduce the idea of parallel EDAs with the migration of the probability models built or the migration of individuals.

In [66], Hiroyaso et al. presented a distributed probabilistic model-building genetic algorithm (DPMBGA). The correlation between the design variables was handled using the Principal Component Analysis (PCA). The authors used the island model where the migration occurred in a

directed ring topology. The migrated individuals were randomly chosen and used to replace the worst individuals in the next sub-population. The authors concluded that using PCA is only useful when dealing with problems in which the design variables are correlated. Experiments also showed that using PCA in half of the sub-populations only provided the best results.

Ahn et al. [67] introduced a basic framework for implementing a parallel EDA and applied it using PBIL. Each island had a resident probability distribution vector (rPV) that estimated the distribution of the promising resident individuals. At every communication step, each island received the immigrant PVs (iPV) from the neighboring islands. The evolution of each island proceeded through three different phases: the *generation phase*, the *selection phase* and the *update phase*. In the generation phase, each island generated three types of individuals, namely, the resident individuals created by rPV, the immigrant individuals created by iPV and crossbred individuals resultant from the crossover of rPV and iPV. In the selection phase, the best individuals were selected from the whole population in a proportionate approach. Finally, in the update phase, the selected individuals helped in updating the different PVs. This framework was used in implementing a discrete parallel EDA based on PBIL referred to as P^2BIL. The introduced approach was found to produce results that are competitive with multiple-deme parallel GAs.

In [68], de la Ossa et al. proposed an island EDA model with the migration of univariate distributions to solve combinatorial problems. Each island adopted the Univariate Marginal Distribution Algorithm (UMDA) [69]. The migrated information between the cooperating islands was a tuple $< M, f >$, where M is the probability model and f is the average fitness of the best 10% individuals of the population. When an island receives an immigrant model M_i, it gets combined with the resident one M_r using the formula below:

$$M_r = \beta M_r + (1 - \beta) * M_i \tag{5.1}$$

where β was set to be 0.9. The authors also proposed an adaptive approach for setting this parameter as follows (for a maximization problem):

$$\beta = \begin{cases} \frac{f_r}{f_i + f_r} & \texttt{if} \quad f_i \geq f_r \\ 0.9 & otherwise \end{cases} \tag{5.2}$$

where f_r and f_i are the average fitness related to the resident and immigrant models respectively. The authors came to the conclusion that migrating a probability model generally gives better results than migrating a group of individuals.

In [64], the same authors extended the application of island-based parallel EDAs to continuous domains. The authors experimented with islands that either adopt $UMDA$ or $EMNA_{GLOBAL}$ [55],

where the latter is used to capture multivariate dependencies. The normal distribution was used to model the promising individuals. Instead of the previous combination model proposed in [68], mixture models were used instead, allowing the combination of single distributions into a joint model. The mixture model was performed as shown based on a coefficient β adaptively set as in equation 5.2.

$$Individual = \begin{cases} sample(M_r) & \texttt{if} \quad random(0,1) < \beta \\ sample(M_i) & otherwise \end{cases} \tag{5.3}$$

The experiments showed that the parallelization was more beneficial when using $UMDA$. When using $EMNA_{GLOBAL}$, the islands required huge populations to correctly model the distribution resulting in a performance deterioration. It was also shown that the migration of a probability model is better than the migration of individuals especially when setting β adaptively.

Madera et al. [70] proposed the use of a distributed version of EDA (dEDA) and applied it to both combinatorial and numerical problems. They used the island model in which each processor executes a $UMDA$ algorithm exchanging information with other processors according to a certain migration policy. Information exchange was applied by selecting the best individuals in one population and replacing the worst individuals in another. The experiments showed that the distributed model was able to solve problems of considerable complexity using a suitable configuration of the migration parameters. The authors also introduced the idea of implementing a *heterogeneous* system where different processors execute different algorithms. However, this approach was not implemented.

In [71,72], Schwarz et al. proposed the use of a parallel bivariate marginal distribution algorithm (BMDA). The island model was used in a directed ring topology. The authors proposed two approaches for combining the immigrant and the resident models. One approach was the *mixed learning of the dependency graphs* experimenting with both the *max* and the *random* operators. The other approach was the *adaptive learning of dependency graphs* employing equations similar to 5.1 and 5.2. The authors reached the conclusion that the migration of probability models with adaptation can significantly improve the performance over the migration of individuals. They also found the sequential BMDA to produce competitive results with the adaptive parallel version but with increased time complexity.

5.2 Proposed Model

In this chapter, we propose the use of a heterogeneous cooperative particle swarm optimizer that is based on probability models exchange. The model uses swarms that employ two different PSO

and EDA hybrids.

5.2.1 Cooperative Swarms

In the hybrid model shown in Figure 5.1, one swarm is using the PSO_Bounds algorithm while the other uses the EDA-PSO approach. At every communication step, each swarm sends its resident model to the other swarm along with the average fitness of the best half of its individuals while receiving the same kind of information back. For the PSO_Bounds swarm, the model sent is a vector containing the lower bound, the higher bound, and the probability of the value being in the higher half for all the dimensions as shown below, where n is the problem size.

$$\boldsymbol{M_{PSO\_Bounds}} =< (a_1, b_1, h_1), (a_2, b_2, h_2), ..., (a_n, b_n, h_n) > \tag{5.4}$$

On the other hand, the EDA-PSO model received is the mean and standard deviation of the different normal distributions used to sample the different dimensions in the other swarm.

$$\boldsymbol{M_{EDA-PSO}} =< (\mu_1, \sigma_1), (\mu_2, \sigma_2), ..., (\mu_n, \sigma_n) > \tag{5.5}$$

Figure 5.1: Hybrid cooperative model.

The model is implemented sequentially by performing the steps shown in Algorithm 5.1 and the communication approach adopted by the model is the *synchronous* one

5.2.2 Probability Model Exchange

At every communication step, each swarm has to extract the useful information from the immigrant model. This is done in two steps, namely, *model conversion* and *model combination*.

5.2.2.1 Model Conversion

In the *model conversion* step, each swarm converts the immigrant model to an equivalent model that is in the same form of its resident one. For the PSO_Bounds swarm, the immigrant model is

Algorithm 5.1 The sequential algorithm for the cooperative model.

Require: Max_Function_Evaluations

1: Initialize the two swarms
2: Max_Iterations = $\frac{Max\_Function\_Evaluations}{Num\_Particles}$
3: iter_numer = 1
4: **while** $iter\_number \leq Max\_Iterations$ **do**
5: Update PSO_Bounds
6: Update EDA-PSO swarm
7: **if** Synchronization **then**
8: Exchange probabilistic models information
9: **end if**
10: iter_number = iter_number + 1
11: **end while**
12: **return** min($gbest_1$,$gbest_2$)

converted as shown below:

$$\begin{aligned} a_d &= \mu_d - \gamma * \sigma_d \\ b_d &= \mu_d + \gamma * \sigma_d \\ h_d &= 0.5, \end{aligned} \tag{5.6}$$

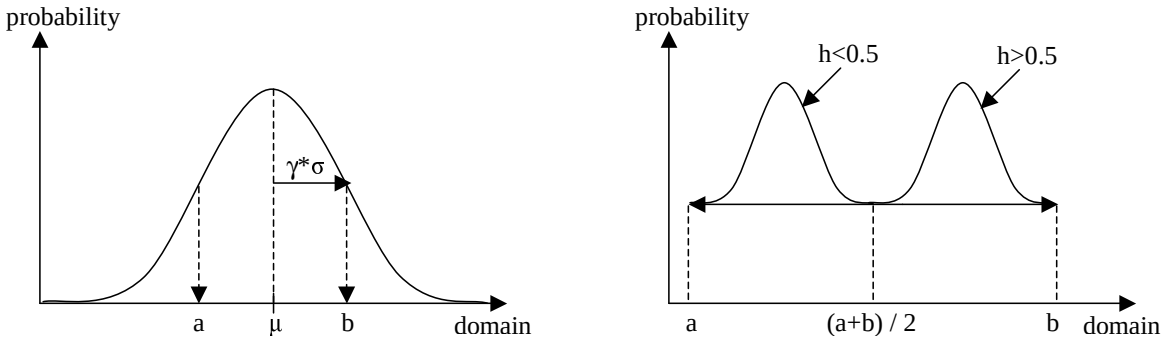
For the EDA-PSO swarm, the process starts by checking the value of h_d to see whether a larger number of particles are in the *low* or the *high* region of the received interval. The process then continues by adjusting the received interval and using it to generate an equivalent Gaussian model according to the following equations, which is equivalent to setting γ equal to 3 in 5.6:

$$\begin{aligned} \mu_d &= \frac{a_d + b_d}{2} \\ \sigma_d &= \frac{b_d - a_d}{6}, \end{aligned} \tag{5.7}$$

Figure 5.2 illustrates the model conversion process carried by the PSO_Bounds and EDA-PSO swarms.

5.2.2.2 Model Combination

After the conversion is done, each swarm will have two models in the same form that it needs to combine. In the PSO_Bounds swarm, the *model combination* step uses the mixture model approach



(a) PSO_Bounds model conversion

(b) EDA-PSO model conversion

Figure 5.2: Probabilistic models conversion.

adopted in [64] on a dimension-by-dimension basis as shown in Algorithm 5.2 for a minimization problem where M_r, M_i, and M_{res} are the resident, immigrant, and resultant models, respectively. $U(0,1)$ is a random number uniformly distributed in the range [0,1]. After this step, the swarm continues with the search process using the resultant combined model.

Algorithm 5.2 The PSO_Bounds swarm models combination function.

Require: $M_{res}, f_{PSO\_Bounds}, M_i, f_{EDA-PSO}$

if $f_{EDA-PSO} < f_{PSO\_Bounds}$ **then**
  $\beta = \frac{f_{EDA-PSO}}{f_{EDA-PSO}+f_{PSO\_Bounds}}$
else
  $\beta = 0.9$
end if
for every dimension d **do**
  if $U(0,1) < \beta$ **then**
    $M_{res}^d = M_r^d$
  else
    $M_{res}^d = M_i^d$
  end if
end for
return M_{res}

For the EDA-PSO swarm, the immigrant model is combined with the resident one following the same approach. The resultant model is then used to generate a number of new particles to replace the worst particles in the swarm. The newly generated particles were chosen to be 10% of the swarm.

5.3 Results and Discussions

5.3.1 Experimental Settings

The parameter settings used for applying the algorithms under study are the same as in Chapter 4 highlighted in Table 4.1. For all experiments, all the particles have been randomly initialized in the specified domain using uniform distribution. The experiments are conducted using the both the classical benchmark functions and the CEC05 benchmark functions f6-f14. In order to constrain the particles movement within the specified domain for the CEC05 functions, any violating particle gets its position randomly re-initialized inside the specified domain. The experiments are run for a problem dimensionality of 10, 30, and 50 with 40 particles in the swarm performing 100000, 100000, and 200000 function evaluations, respectively. The results reported are the averages taken over 30 runs and are the error values $f(x) - f(x*)$, where $x*$ is the global optimum.

In the *model conversion* step of the PSO_Bounds swarm, we set the value of γ to 1, which means that the interval $[a, b]$ should contain 68% of the values that could be generated using the received Gaussian distribution.

5.3.2 Results of the Proposed Model

The model is applied using 40 particles per swarm, where each swarm performs 50000, 50000, and 100000 function evaluations (half the number performed by the individual runs reported in Chapter 4) so as for the whole model to perform the same number of function evaluations as the individual runs. The model adopted different synchronization periods [10, 25, 50, 100, 250, 500] allowing [125, 50, 25, 12, 5, 2] communication stages, respectively.

Table 5.1 and Table 5.2 show the results of applying the proposed cooperative model for problem sizes of 10, 30, and 50. The tables show the best result achieved by the model and the synchronization period at which this result was obtained. The significance column is based on a two sample t-test that is run to verify the results statistical significance. If this column indicates a **Yes**, it means that the null hypothesis is rejected with a 95% confidence level.

The results show that the cooperative model has a generally not to good performance in the unimodal functions (the Spherical, the Rosenbrock and f6). The reason for not providing good results for these functions is due to the poor performance of PSO_Bounds in them. In these functions, PSO_Bounds provides results that are worse than the results of EDA-PSO by orders of magnitude. The results provided by the cooperative model is very promising considering that

Table 5.1: Results of the cooperative model for the classical functions.

Benchmark Function	**Dimensionality**	**Synchronization Period**	**Mean**	**Std.**	**Significantly better than its components**
Spherical	10	10	3.572e-134	1.182e-133	No
Rosenbrock		*50*	*4.562e-02*	*5.539e-02*	*The same*
Griewank		*25*	*4.598e-03*	*6.457e-03*	*The same*
Ackley		*25*	*5.887e-16*	*2.006e-31*	*The same*
Rastrigin		**25**	**1.808e+00**	**1.243e+00**	**Yes**
Spherical	30	10	3.075e-70	5.927e-70	No
Rosenbrock		100	1.876e+00	3.416e-01	No
Griewank		**25**	**0**	**0**	**Yes**
Ackley		**25**	**3.431e-15**	**1.445e-15**	**Yes**
Rastrigin		**25**	**1.343e+01**	**3.561e+00**	**Yes**
Spherical	50	*25*	*5.406e-92*	*2.961e-91*	*The same*
Rosenbrock		*100*	*2.488e+00*	*2.961e-01*	*The same*
Griewank		*500*	*2.465e-04*	*1.350e-03*	*The same*
Ackley		**100**	**4.141e-015**	**0**	**Yes**
Rastrigin		**10**	**2.686e+01**	**6.145e+00**	**Yes**

half of the function evaluations were consumed by the PSO_Bounds component. In most of the multimodal functions, the cooperative model is always able to at least match the solution of the better performing component, if not improving on it.

The experiments are re-run again for all the algorithms using the *lbest* model. Table 5.3 and Table 5.4 show the obtained results. The results show the same trend as the only cases in which the cooperative model is worse than the better component are in the unimodal functions.

In general, when using the *gbest* model, the cooperative approach produced better results than its components in 36% of the studied cases, while maintaining the same quality of solutions as the better component in 42% of the cases, and failing to do even so in the remaining 22%.

When using the *lbest* model, the results are a little better. The cooperative approach produced better results than its components in 47% of the studied cases, while maintaining the same quality of solutions as the better component in 42% of the cases, and failing to do even so in the remaining 11%.

Table 5.2: Results of the cooperative model for the CEC05 functions.

Benchmark Function	Dimensionality	Synchronization Period	Mean	Std.	Significantly better than its components
f6	10	100	1.621e-01	1.048e-01	No
f9		**10**	**2.668e+00**	**1.374e+00**	**Yes**
f10		*10*	*3.488e+00*	*1.341e+00*	*The same*
f11		*10*	*3.263e+00*	*2.010e+00*	*The same*
f12		*100*	*2.564e+03*	*2.402e+03*	*The same*
f13		*10*	*6.377e-01*	*1.984e-01*	*The same*
f14		*10*	*2.438e+00*	*4.711e-01*	*The same*
f6	30	250	3.843e+01	4.076e+01	No
f9		**50**	**2.216e+01**	**4.241e+00**	**Yes**
f10		*10*	*5.568e+01*	*4.620e+01*	*The same*
f11		*10*	*2.610e+01*	*6.380e+00*	*The same*
f12		*100*	*2.088e+05*	*1.683e+05*	*The same*
f13		**50**	**3.301e+00**	**9.085e-01**	**Yes**
f14		*25*	*1.258e+01*	*3.972e-01*	*The same*
f6	50	250	5.813e+01	4.700e+01	No
f9		100	5.250e+01	1.296e+01	No
f10		**10**	**8.699e+01**	**7.753e+01**	**Yes**
f11		**10**	**5.135e+01**	**9.456e+00**	**Yes**
f12		50	1.565e+05	1.858e+05	No
f13		**10**	**6.003e+00**	**1.086e+00**	**Yes**
f14		**25**	**2.204e+01**	**4.483e-01**	**Yes**

5.3.3 Convergence Behavior

The convergence behavior shown in Figure 5.3, Figure 5.4, Figure 5.5, and Figure 5.6 illustrate an interesting feature as the cooperative model always tries to follow the algorithm that performs better on the problem in hand:

- For the Spherical, the Rosenbrock, the Griewank, and f6, the cooperative model is always following the behavior of EDA-PSO as it provides much better results than PSO_Bounds.
- For the Rastrigin, f9, f10, f12, and f13, the model starts by following EDA-PSO as it has a faster convergence in the beginning, once EDA-PSO gets to the part of the search where it stagnates, the model switches to follow the behavior of PSO_Bounds.

Table 5.3: Results of all the cooperative model the *lbest* model for the classical functions.

Benchmark Function	Dimensionality	Synchronization Period	Mean	Std.	Significantly better than its components
Spherical	10	*500*	**2.099e-114**	**6.131e-114**	*The same*
Rosenbrock		50	1.415e-01	5.939e-02	No
Griewank		*50*	**4.600e-03**	**5.778e-03**	*The same*
Ackley		**10**	**5.888e-16**	**2.006e-31**	**Yes**
Rastrigin		**100**	**9.464e-01**	**7.781e-01**	**Yes**
Spherical	30	**250**	4.109e-61	2.135e-60	No
Rosenbrock		*100*	**2.236e+00**	**2.643e-01**	*The same*
Griewank		**50**	**0**	**0**	**Yes**
Ackley		**50**	**2.957e-15**	**1.703e-15**	**Yes**
Rastrigin		**10**	**1.092e+01**	**3.817e+00**	**Yes**
Spherical	50	**50**	**2.372e-59**	**1.235e-56**	**Yes**
Rosenbrock		*500*	**4.420e+00**	**4.409e+00**	*The same*
Griewank		*10*	**2.465e-04**	**1.350e-03**	*The same*
Ackley		**25**	**4.144e-15**	**0**	**Yes**
Rastrigin		**10**	**2.448e+01**	**6.685e+00**	**Yes**

- For f11 and f14, the model follows PSO_Bounds from start to finish as it has the better performance during the whole search.

5.3.4 Synchronization Period Effect

The effect of changing the synchronization period on the model performance is shown in Figure 5.7, Figure 5.8, and Figure 5.9. The way the performance changes seems to be dependent on the function under study and is not affected by the problem size.

In general, increasing the synchronization period may cause the performance of the model to improve up to a certain limit after which the solution starts to deteriorate, which is clear for f6, f10, f11, f12, and f14.

For f9 and f13, there is a different behavior in which the solution obtained by the model tends to increase, decrease and finally increase again as the synchronization period is increased. Again, this behavior is observed for all problem sizes.

Table 5.4: Results of the cooperative model using the *lbest* model for the CEC05 functions.

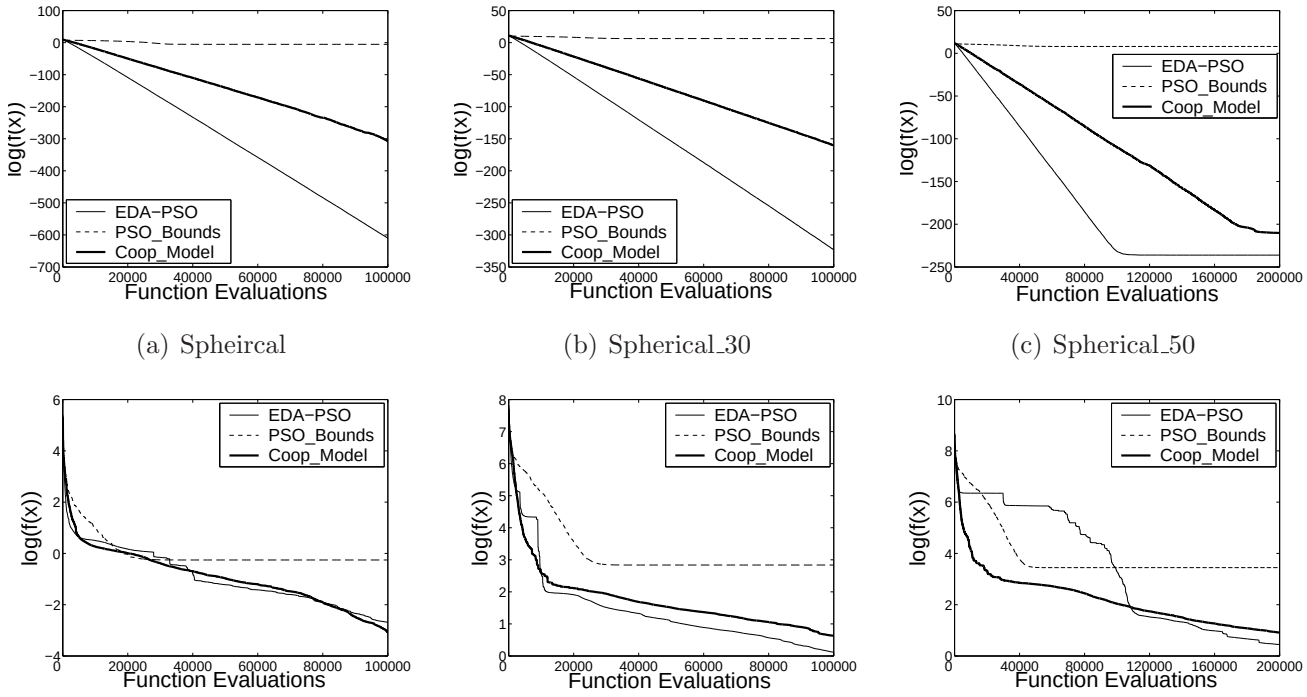
Benchmark Function	Dimensionality	Synchronization Period	Mean	Std.	Significantly better than its components
f6	10	500	1.479e+00	4.890e-01	No
f9		*500*	**1.090e+00**	**1.198e+00**	*The same*
f10		**10**	**2.913e+00**	**1.756e+00**	**Yes**
f11		*10*	**5.033e+00**	**1.299e+00**	*The same*
f12		*50*	**6.089e+03**	**2.878e+03**	*The same*
f13		**25**	**5.488e-01**	**1.447e-01**	**Yes**
f14		*10*	**2.631e+00**	**2.831e-01**	*The same*
f6	30	*500*	**5.701e+01**	**4.431e+01**	*The same*
f9		100	**2.262e+01**	**6.062e+00**	*The same*
f10		**25**	**6.392e+01**	**2.773e+01**	**Yes**
f11		*10*	**2.997e+01**	**2.670e+00**	*The same*
f12		*10*	**4.680e+05**	**1.040e+05**	*The same*
f13		**10**	**3.831e+00**	**7.884e-01**	**Yes**
f14		**25**	**1.270e+01**	**2.383e-01**	**Yes**
f6	50	500	1.106e+02	7.083e+01	No
f9		**100**	**4.897e+01**	**8.206e+00**	**Yes**
f10		**50**	**1.387e+02**	**7.613e+01**	**Yes**
f11		*10*	**5.967e+01**	**4.527e+00**	*The same*
f12		*50*	**1.697e+06**	**4.267e+05**	*The same*
f13		**10**	**6.614e+00**	**1.959e+00**	**Yes**
f14		**10**	**2.228e+01**	**3.520e-01**	**Yes**

5.3.5 Exchanging Particles

Previous research in [64, 68, 71] all reached a similar conclusion where the migration of probability models can provide better results than the migration of individuals. In this section, we investigate this idea by applying the cooperative model while sharing a group of particles. The migration approach adopted is selecting the best 10% particles in one swarm to replace the worst 10% particles in the other swarm.

The best results achieved by the cooperative model are shown in Table 5.5. The results show that there is no migration scheme that outperforms the other in all benchmark functions. While migrating particles significantly improved the results for both f6 and f12, the results deteriorated for f9 and f10. For the remaining two functions, both approaches produced similar results.

Figure 5.10 and Figure 5.11 show the performance of the cooperative model across different



(a) Spheircal (b) Spherical_30 (c) Spherical_50

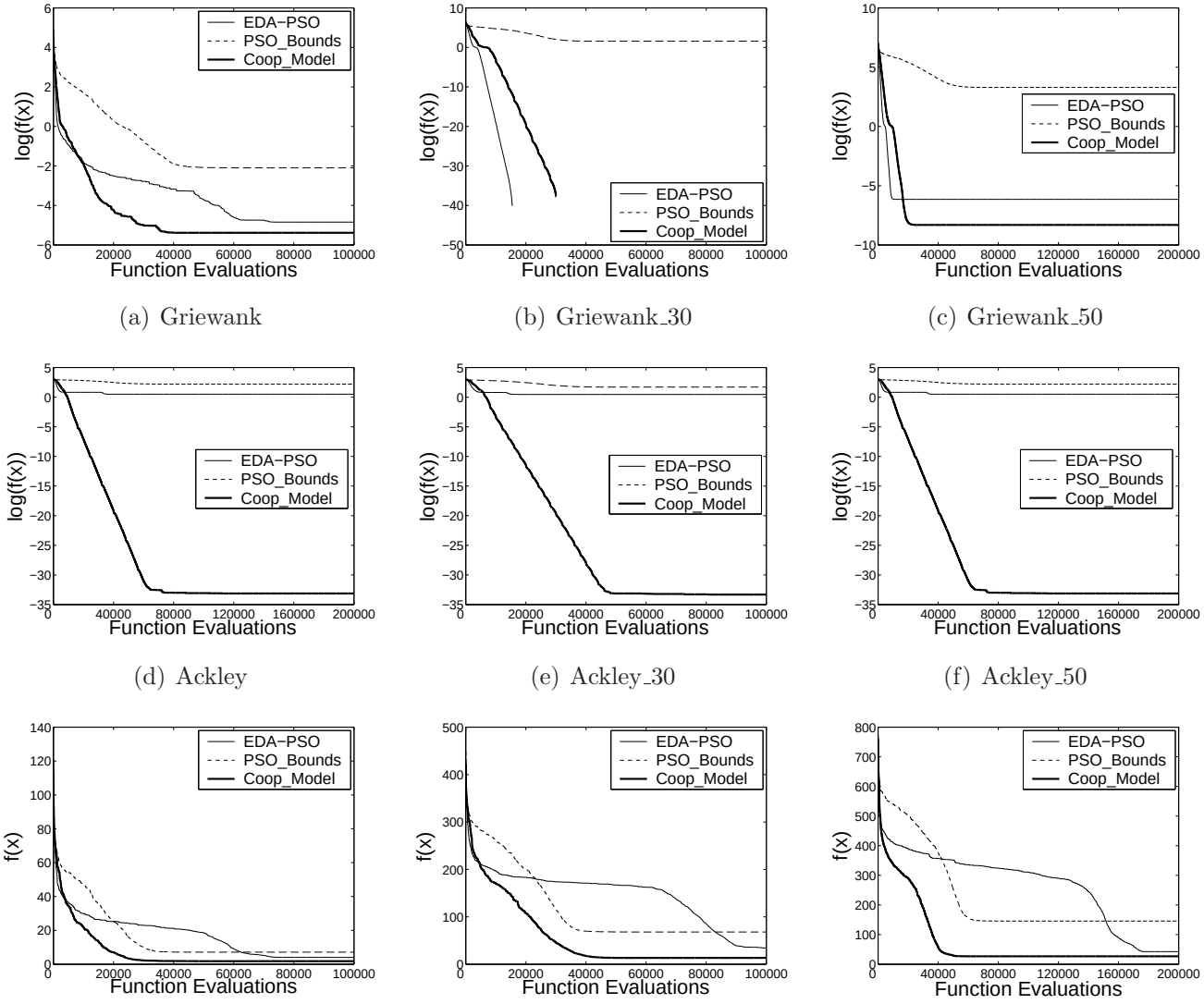
(d) Rosenbrock (e) Rosenbrock_30 (f) Rosenbrock_50

Figure 5.3: Convergence behavior of the three algorithms for the unimodal classical functions.

Table 5.5: Results of cooperative model adopting particles migration for a dimensionality of 10.

Benchmark Function	**Syncronization Period**	**Mean**	**Std.**	**Compared to probabilistic models migration**
f6	**10**	**5.77e-02**	**1.91e-01**	**Better**
f9	100	3.65e+00	1.60e+00	Worse
f10	250	6.53e+00	2.79e+00	Worse
f11	25	3.85e+00	1.96e+00	Worse
f12	**10**	**3.30e+02**	**6.16e+02**	**Better**
f13	*100*	*6.54e-01*	*1.83e-01*	*The same*
f14	*250*	*2.54e+00*	*5.054e-01*	*The same*

synchronization periods when it adopts the two migration schemes. Again, the results show that there is no migration scheme that outperforms the other on all the benchmark functions. In Spherical, Griewank, Rastrigin, f9, and f10 migrating a probability model always gives better results than migrating particles. While in f12, migrating particles significantly outperforms migrating a probability model, which is also maintained at higher dimensionalities as shown in Figure 5.12. In f6, f11, and f13, no approach outperforms the other on all the synchronization periods.



(a) Griewank (b) Griewank_30 (c) Griewank_50

(d) Ackley (e) Ackley_30 (f) Ackley_50

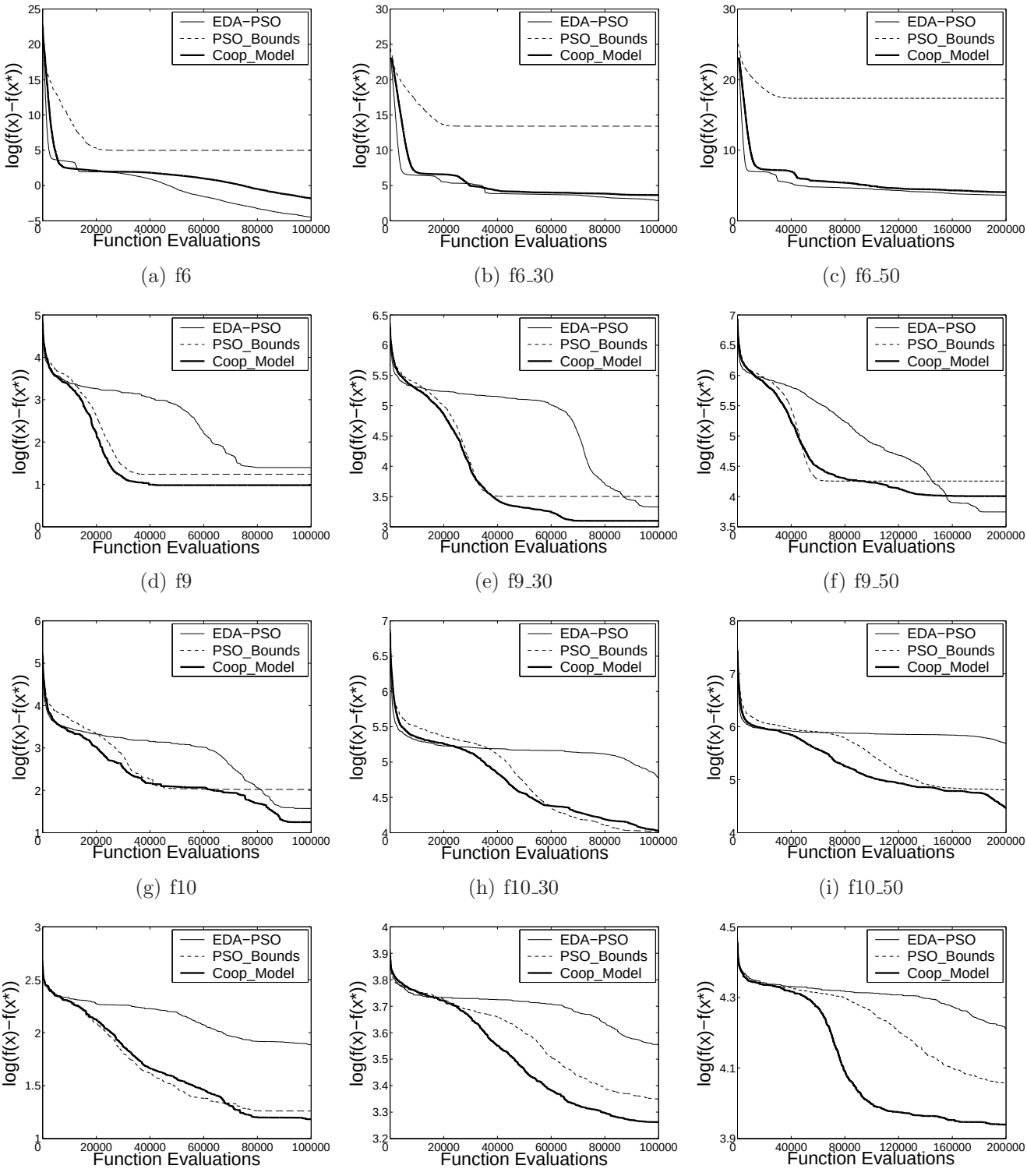
(g) Rastrigin (h) Rastrigin_30 (i) Rastrigin_50

Figure 5.4: Convergence behavior of the three algorithms for the multimodal classical functions.

5.3.6 A Simple Adaptive Version

As noted before, the cooperative model is not able to produce better results than its components if one of them has a poor performance in the function under study. One approach to improve the cooperative model performance is to increase the effect of the component performing better on the function being optimized. This is investigated in this section by implementing a simple adaptive version of the cooperative model. The adaptation could be done by:

- Changing the amount of information exchanged during the search.
- Changing the number of function evaluations performed by each component.



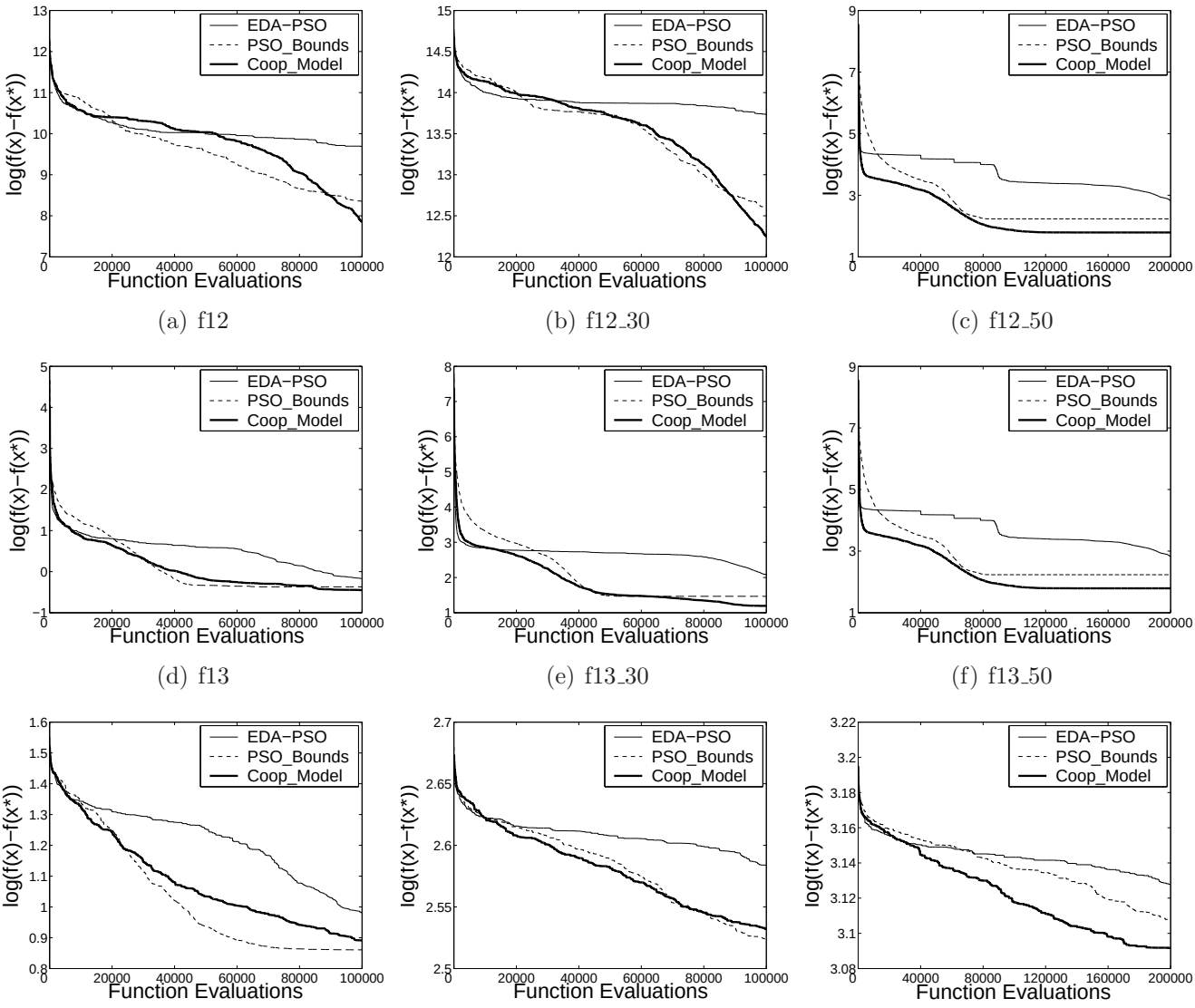
(a) f6 (b) f6_30 (c) f6_50

(d) f9 (e) f9_30 (f) f9_50

(g) f10 (h) f10_30 (i) f10_50

(j) f11 (k) f11_30 (l) f11_50

Figure 5.5: Convergence behavior of the three algorithms for the CEC05 benchmark functions.



(a) f12 (b) f12_30 (c) f12_50

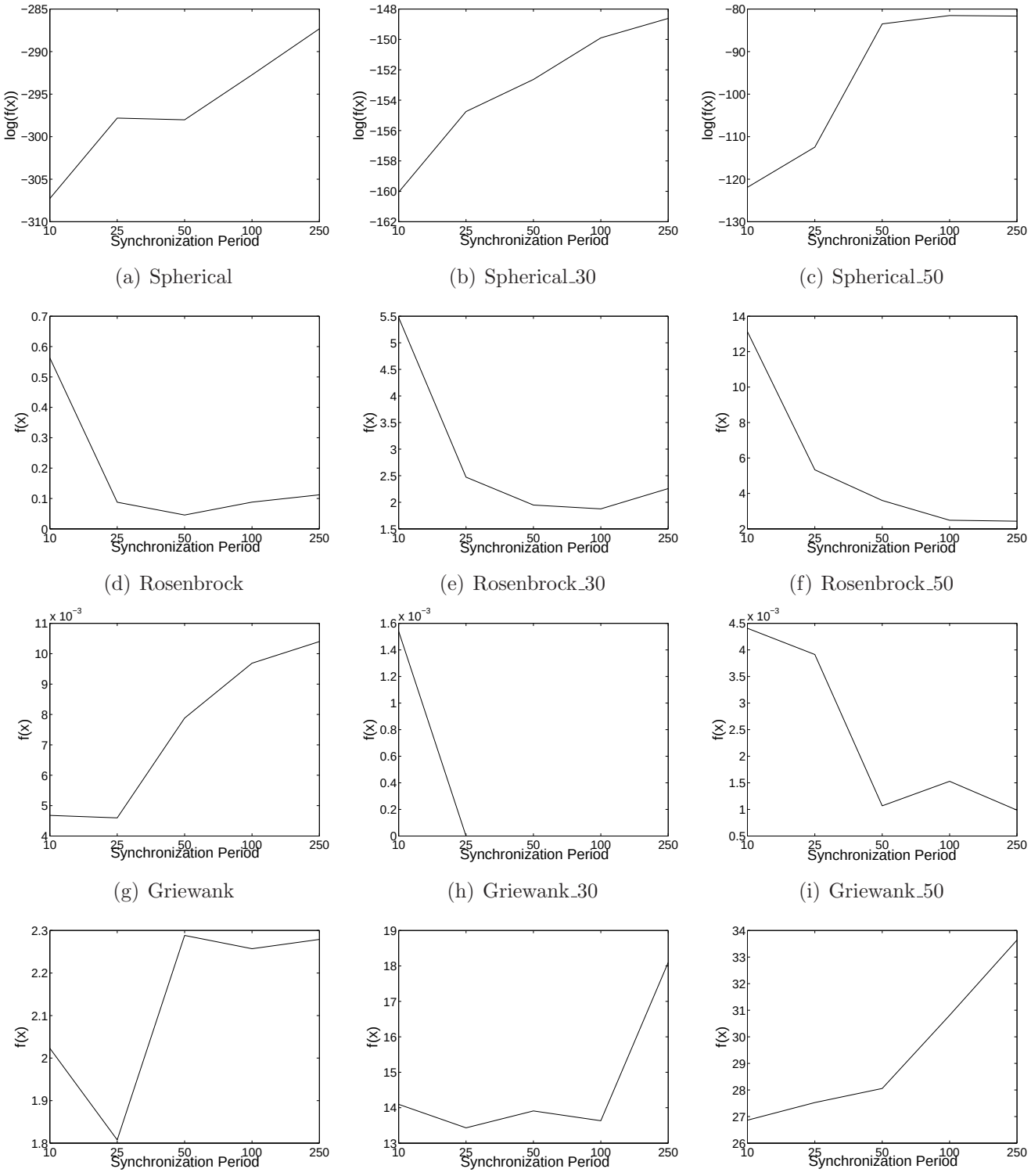
(d) f13 (e) f13_30 (f) f13_50

(g) f14 (h) f14_30 (i) f14_50

Figure 5.6: Convergence behavior of the three algorithms for the CEC05 benchmark functions, contd.

- Changing both.

In this work, the first approach is adopted. This is done by observing the performance of both components in every iteration during the search process. Two counters are used, one for each component, and when a certain component has a better performance during any given iteration, its counter gets incremented. To accommodate the fact that these components have different behaviors during the search, a *sliding window* approach is taken by resetting these counters every 50 iterations. At the end of every sliding window, the percentage of number of iterations in which the PSO_Bounds had the better performance is calculated. This percentage is used to control the



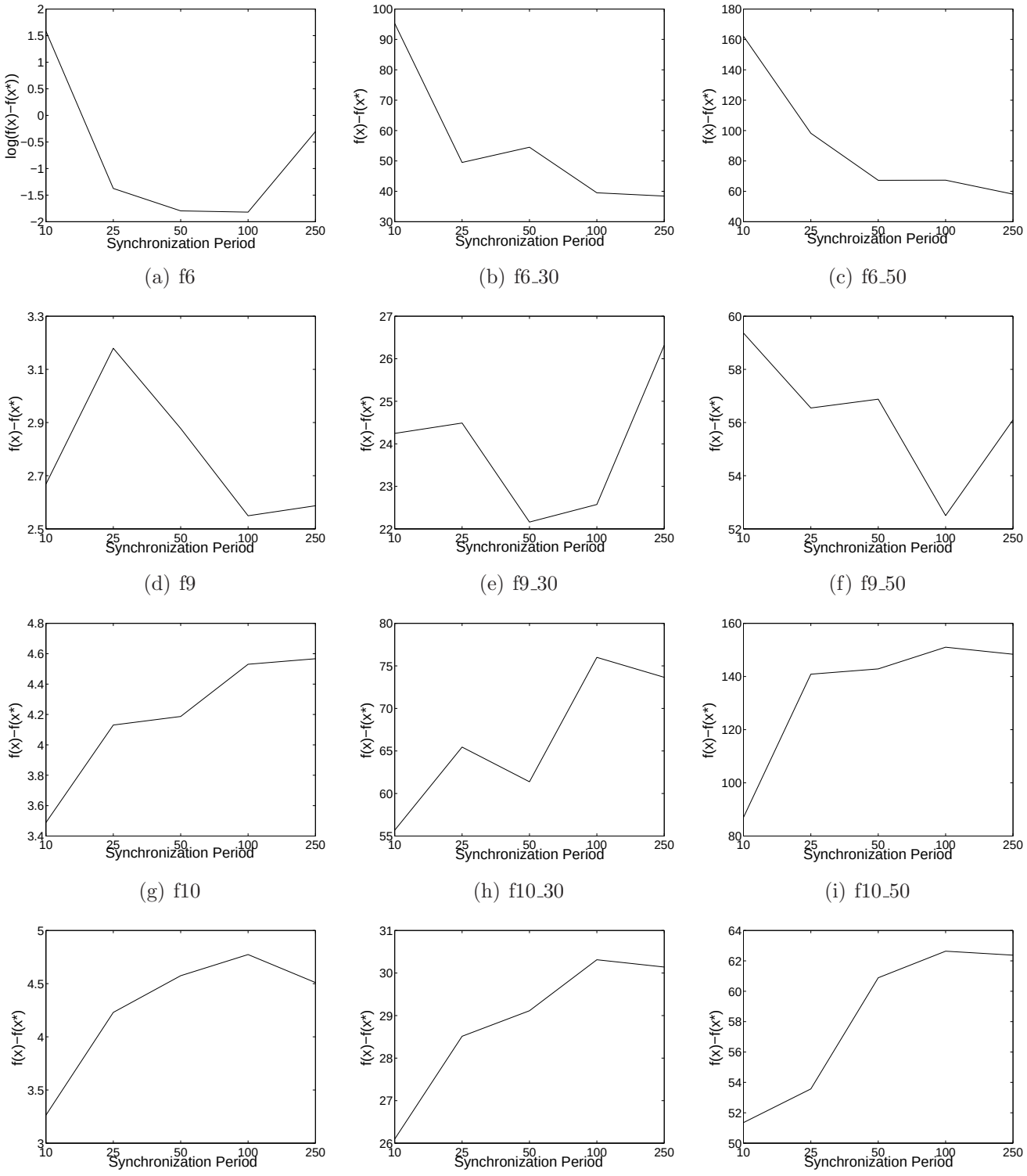
(a) Spherical (b) Spherical_30 (c) Spherical_50

(d) Rosenbrock (e) Rosenbrock_30 (f) Rosenbrock_50

(g) Griewank (h) Griewank_30 (i) Griewank_50

(j) Rastrigin (k) Rastrigin_30 (l) Rastrigin_50

Figure 5.7: Synchronization period effect for the cooperative model.



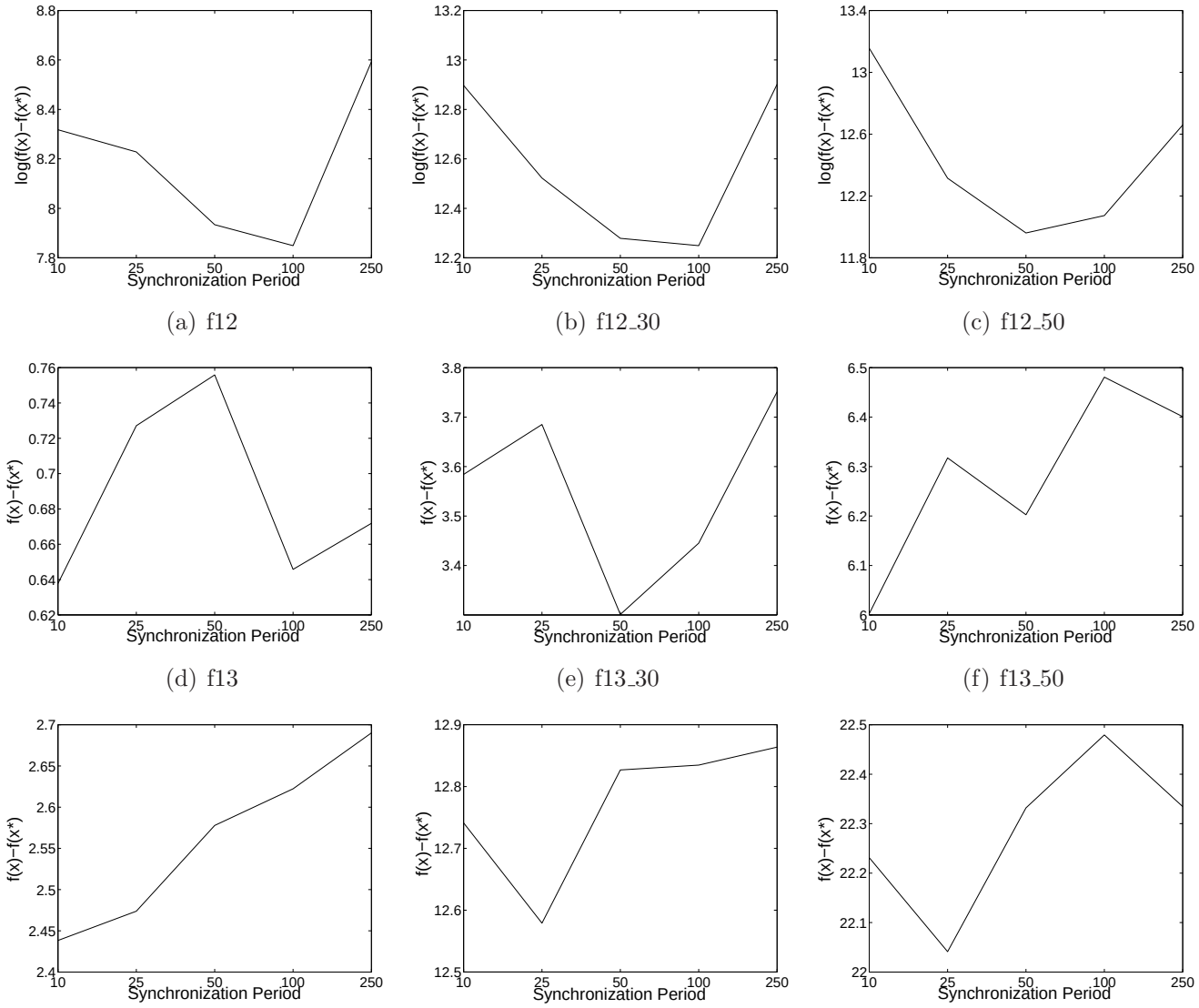
(a) f6 (b) f6_30 (c) f6_50

(d) f9 (e) f9_30 (f) f9_50

(g) f10 (h) f10_30 (i) f10_50

(j) f11 (k) f11_30 (l) f11_50

Figure 5.8: Synchronization period effect for the cooperative model, contd.



(a) f12 (b) f12_30 (c) f12_50

(d) f13 (e) f13_30 (f) f13_50

(g) f14 (h) f14_30 (i) f14_50

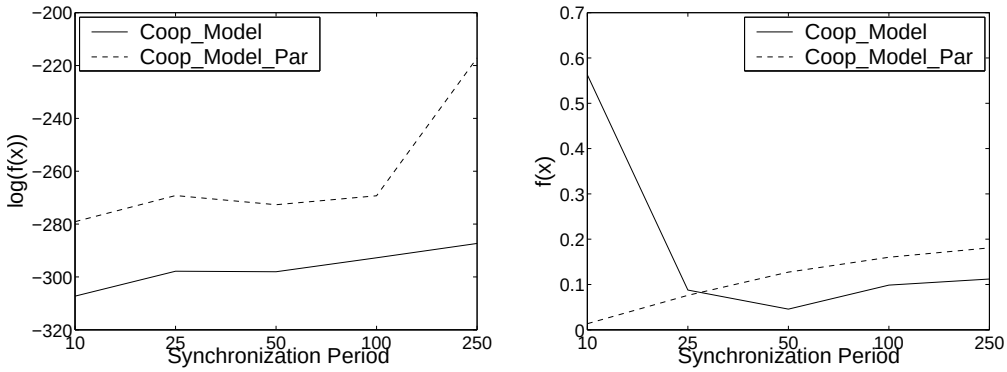
Figure 5.9: Synchronization period effect for the cooperative model, contd.

amount of information flow between the two swarms.

To increase the PSO_Bounds swarm effect, the number of particles replaced in the EDA-PSO swarm is increased. On the other hand, to increase the EDA-PSO swarm effect, the calculation of β is the PSO_Bounds swarm is changed as follows:

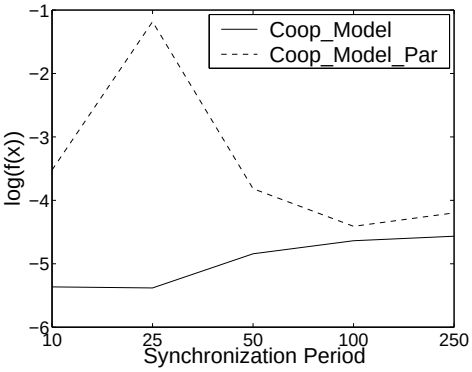
$$\beta = \begin{cases} factor * \frac{f_{EDA-PSO}}{f_{EDA-PSO}+f_{PSO\_Bounds}} & \texttt{if} \quad f_{EDA-PSO} < f_{PSO\_Bounds} \\ factor * 0.9 & otherwise \end{cases} \tag{5.8}$$

where $factor$ controls the influence of the EDA-PSO swarm. Normally $factor$ is set to 1, but when it decreases this increases the influence of the EDA-PSO swarm.

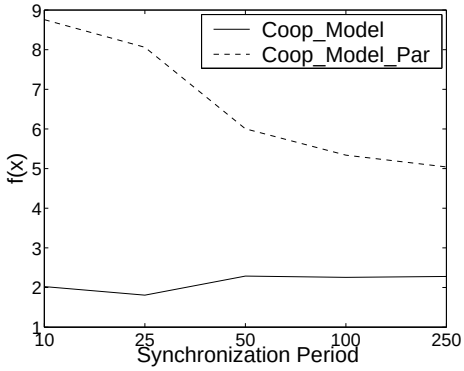


(a) Spherical

(b) Rosenbrock



(c) Griewank



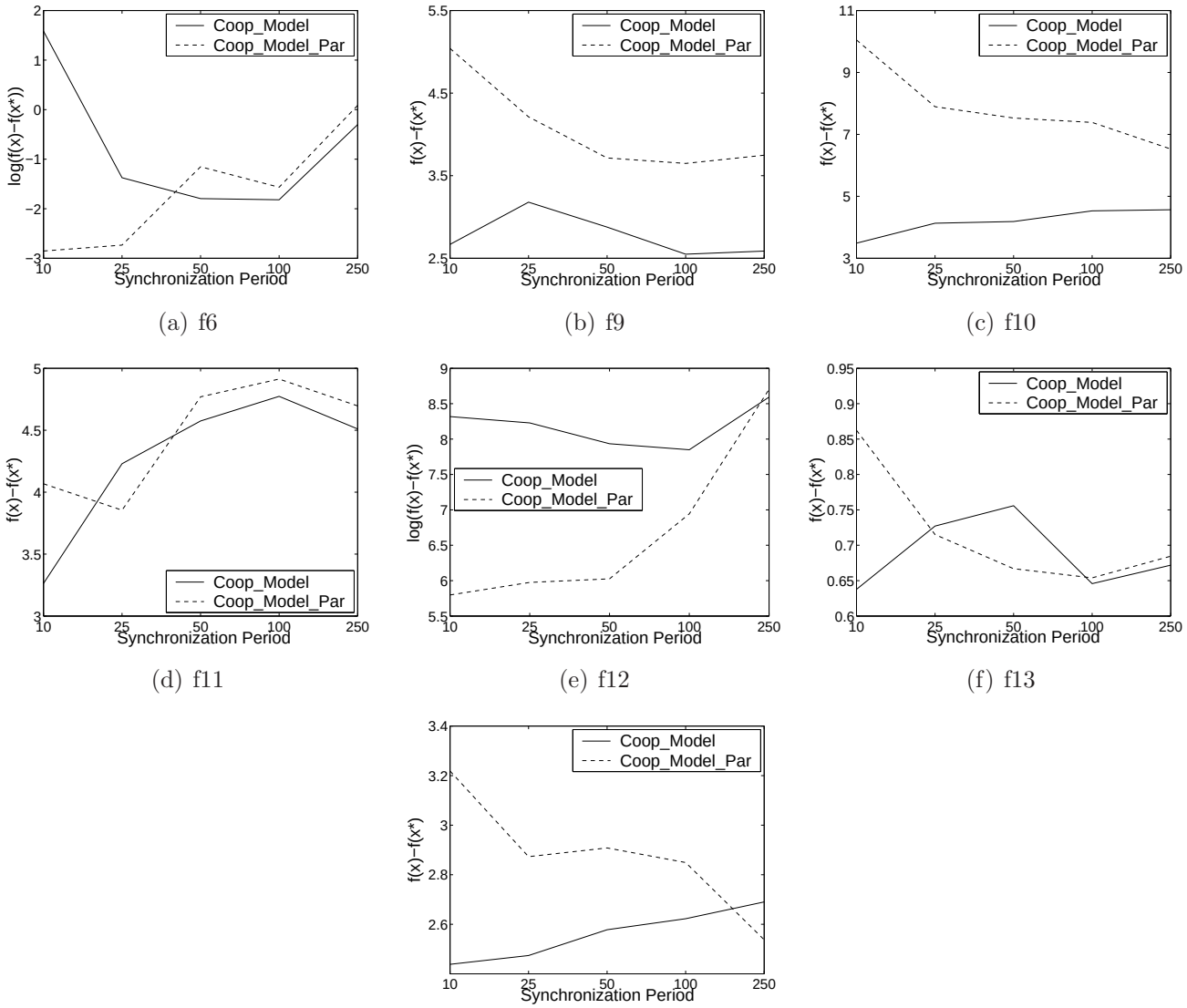
(d) Rastrigin

Figure 5.10: Comparing probabilistic model migration vs. particles migration for the classical functions for a dimensionality of 10.

Algorithm 5.3 shows the steps taken by the adaptive model at the end of each sliding window where α is the percentage of the PSO_Bounds model performing better during the previous window period. When α is equal to 0.5, the normal information flow is taken. When PSO_Bounds is the better performer, the number of replaced particles is increased from 4(10%) to 8(20%). If the EDA-PSO has the better performance, the number is decreased to 2(5%) to minimize the PSO_Bounds effect and $factor$ is also decreased to 0.25 to increase the EDA-PSO effect.

Table 5.6 shows the results of the adaptive *gbest* model across the different dimensions for the classical functions. The results show that the adaptive version has a significantly better performance over the non-adaptive version and it's able to reach the global optimum for different cases.

Table 5.7 shows the results of the adaptive model across the different dimensions for the CEC05 benchmark functions. The results show that the using the adaptive approach enabled the cooperative model to produce better results than its components for more cases than the non-adaptive version.



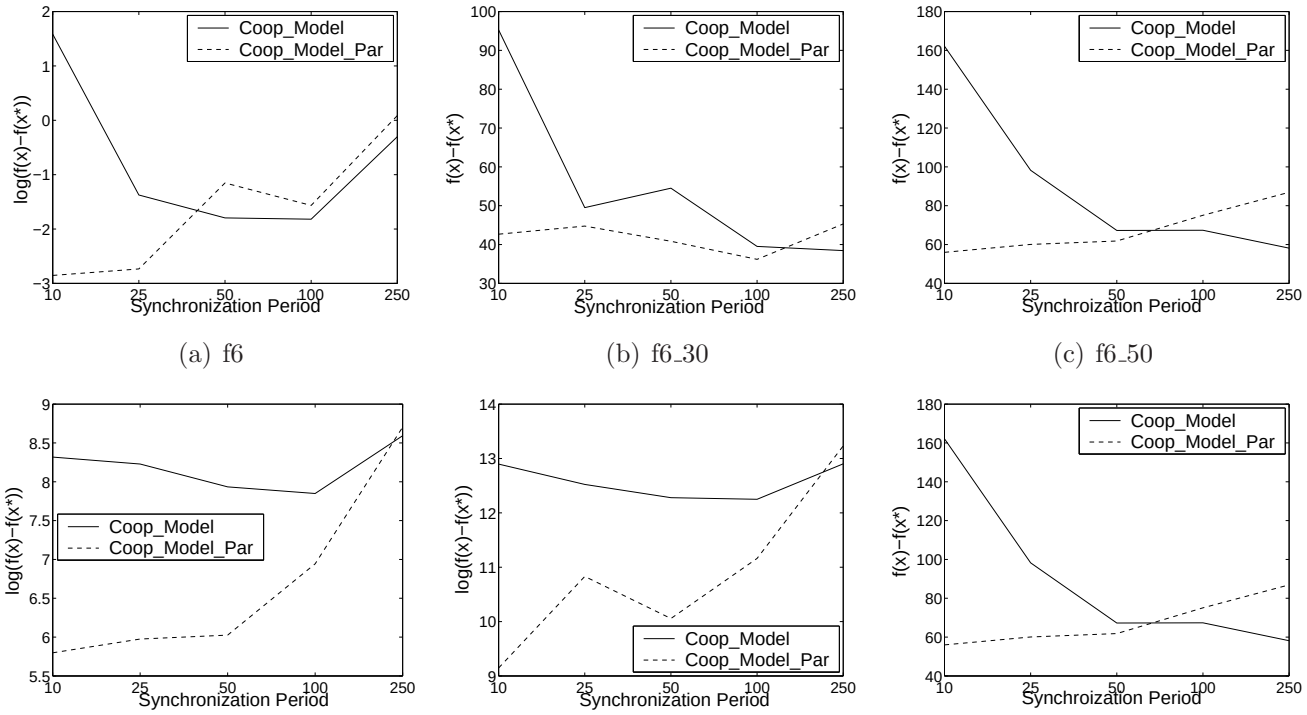
(a) f6 (b) f9 (c) f10

(d) f11 (e) f12 (f) f13

(g) f14

Figure 5.11: Comparing probabilistic model migration vs. particles migration for the CEC05 benchmark functions for a dimensionality of 10.

Figure 5.13 shows the comparison between the two approaches when applied using both the *gbest* and *lbest* models based on their success rate in producing better results than their components. It shows that adapting the information flow between the two components has resulted in increasing the number of cases in which the cooperative model improves the results. It also shows that the adaptive model fails to even match the result of the better component in only less than 10% of the studied cases (most of which are unimodal functions). The results also indicate that the performance of the adaptive model is robust against changing the underlying population topology.



(a) f6 (b) f6_30 (c) f6_50

(d) f12 (e) f12_30 (f) f12_50

Figure 5.12: Comparing probabilistic model migration vs. particles migration for all dimensionalities.

Algorithm 5.3 The adaptive information flow algorithm.

Require: PSO_Bounds_Counter, EDA-PSO_Counter

$\alpha = \frac{PSO\_Bounds\_Counter}{PSO\_Bounds\_Counter + EDA - PSO\_Counter}$
if $\alpha = 0.5$ **then**
    number_of_particles_to_replace = 4
    factor = 1
else if $\alpha > 0.5$ **then**
    number_of_particles_to_replace = 8
    factor = 1
else
    number_of_particles_to_replace = 2
    factor = 0.25
end if
PSO_Bounds_Counter = 0
EDA-PSO_Counter = 0

Table 5.6: Results of the adaptive *gbest* cooperative model for the classical functions.

Benchmark Function	Dimensionality	Synchronization Period	Mean	Std.	Significantly better than its components
Spherical	10	**10**	**0**	**0**	**Yes**
Rosenbrock		*10*	*9.492e-02*	*8.388e-02*	*The same*
Griewank		*10*	*4.435e-03*	*6.046e-03*	*The same*
Ackley		**10**	**0**	**0**	**Yes**
Rastrigin		**10**	**1.556e+00**	**1.034e+00**	**Yes**
Spherical	30	**10**	**0**	**0**	**Yes**
Rosenbrock		25	1.888e+00	3.198e-01	No
Griewank		**10**	**0**	**0**	**Yes**
Ackley		**10**	**0**	**0**	**Yes**
Rastrigin		**10**	**1.267e+01**	**3.396e+00**	**Yes**
Spherical	50	**10**	**0**	**0**	**Yes**
Rosenbrock		*100*	*2.587e+00*	*8.365e-01*	*The same*
Griewank		*500*	*4.105e-04*	*2.249e-03*	*The same*
Ackley		**25**	**0**	**0**	**Yes**
Rastrigin		**10**	**2.806e+01**	**9.372e+00**	**Yes**

5.4 Comparison with other PSO cooperative models

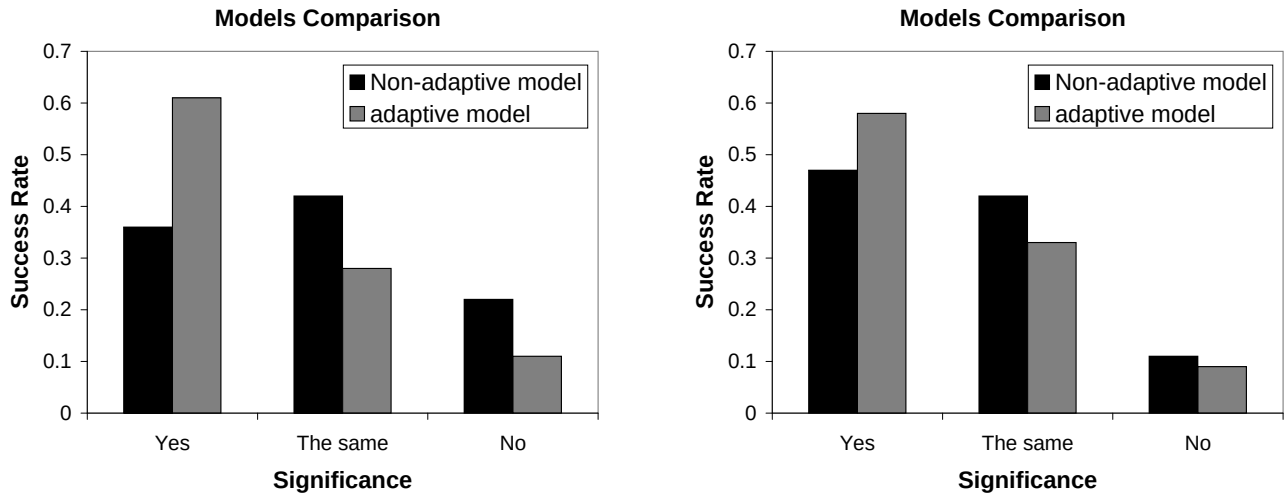
To test how the proposed adaptive cooperative model performs in comparison with other state-of-the-art PSO cooperative algorithms, Table 5.8 shows the comparison between our results and the following approaches:

- **CPSO_S** [9] : A cooperative PSO approach where each dimension is being optimized by a separate swarm. The approach uses 10 particles per swarm as this was shown in [9] to be the best.
- **DMS-L-PSO** [21, 22]: A a dynamic *lbest* multi-swarm approach in which particles get randomly and continuously assigned to different swarms. The approach is also combined with the Quasi-Newton method to improve its local search ability. The approach uses 20 swarms and 3 particles per swarm. These swarms get randomly re-constructed every 5 iterations and the Quasi-Newton method is performed on the best 20% particles *pbests* every 100 iterations.
- **TRIBES-D** [73]: A parameter free PSO having multiple swarms, referred to as $TRIBES$. The tribes share information among them and have the capability to destroy bad particles and/or randomly generate new ones to form a new tribe. The source code is available at [74].

Table 5.7: Results of the adaptive *gbest* cooperative model for the CEC05 benchmark functions.

Benchmark Function	**Dimensionality**	**Synchronization Period**	**Mean**	**Std.**	**Significantly better than its components**
f6	10	**10**	**1.529e-01**	**1.907e-02**	No
f9		**25**	**2.361e+00**	**1.151e+00**	**Yes**
f10		**10**	**2.952e+00**	**1.578e+00**	**Yes**
f11		*10*	*3.311e+00*	*1.962e+00*	*The same*
f12		**100**	**1.10e+03**	**1.229e+03**	**Yes**
f13		*25*	*6.336e-01*	*2.005e-01*	*The same*
f14		*10*	*2.424e+00*	*3.885e-01*	*The same*
f6	30	*25*	*2.762e+01*	*2.778e+01*	*The same*
f9		10	**2.302e+01**	**5.410e+00**	**Yes**
f10		**25**	**2.922e+01**	**1.931e+01**	**Yes**
f11		*50*	*2.737e+01*	*6.600e+00*	*The same*
f12		**50**	**1.500e+05**	**1.726e+05**	**Yes**
f13		**25**	**3.095e+00**	**7.375e-01**	**Yes**
f14		*10*	*1.259e+01*	*4.643e-01*	*The same*
f6	50	25	5.582e+01	3.082e+01	No
f9		10	5.479e+01	8.446e+00	No
f10		**10**	**6.204e+01**	**2.611e+01**	**Yes**
f11		**50**	**4.921e+01**	**8.997e+00**	**Yes**
f12		**50**	**7.328e+04**	**6.118e+04**	**Yes**
f13		**10**	**5.518e+00**	**1.274e+00**	**Yes**
f14		**25**	**2.197e+01**	**4.094e-01**	**Yes**

Table 5.8 and Table 5.9 show the complete results of theses algorithms for the classical functions and the CEC05 benchmark functions in all dimensions. Table 5.10 summarizes this comparison showing the number of cases in which each algorithm was the best out of the 15 cases (5 functions in 3 dimensions) in the classical functions and the 21 cases (7 functions in 3 dimensions) in the CEC05 benchmark functions. The comparison also shows the function in which each algorithm provided the best result in all dimensions. The results show that the adaptive_*gbest* model is very competitive with the state-of-the-art PSO cooperative algorithms and that there is no algorithm that outperforms all the others on more than two functions.



(a) gbest model

(b) lbest model

Figure 5.13: Non-Adaptive vs. Adaptive Model Performance.

Table 5.8: Results of all the algorithms for the classical functions

Benc. Func.	**Dim.**	Adaptive_*gbest*		DMS-L-PSO		CPSO_S		TRIBES-D	
		Mean	**Std.**	**Mean**	**Std.**	**Mean**	**Std.**	**Mean**	**Std.**
Spherical	10	**0**	**0**	3.45e-59	5.72e-59	1.33e-155	5.35e-155	**0**	**0**
Rosenbrock		9.49e-02	8.39e-02	**6.10e-06**	**2.27e-05**	9.76e-03	1.85e-02	1.73e-03	7.90e-03
Griewank		4.44e-03	6.05e-03	**0**	**0**	4.89e-02	2.46e-02	4.30e-02	2.45e-02
Ackley		**0**	**0**	2.31e-15	1.08e-15	1.08e-14	4.25e-15	**0**	**0**
Rastrigin		1.56+e00	1.03e+00	**0**	**0**	**0**	**0**	**0**	**0**
Spherical	30	**0**	**0**	8.98e-27	1.08e-26	4.94e-49	1.48e-48	**0**	**0**
Rosenbrock		1.89e+00	3.20e-01	**1.72e-07**	**2.66e-07**	3.05e-01	1.05e-01	6.76e-01	3.26e-01
Griewank		**0**	**0**	1.48e-16	1.10e-16	2.46e-02	1.88e-02	4.82e-02	4.87e-02
Ackley		**0**	**0**	1.12e-12	2.10e-12	4.05e-14	1.24e-14	3.92e-04	8.46e-04
Rastrigin		1.27e+01	3.40e+00	1.78e+01	4.06e+00	**0**	**0**	2.29e+00	1.55e+00
Spherical	50	**0**	**0**	1.36e-29	1.15e-29	4.56e-57	2.17e-56	**0**	**0**
Rosenbrock		2.59e+00	8.36e-01	5.31e-01	1.46e-01	**3.64e-01**	**1.32e-01**	9.10e-01	3.04e-01
Griewank		4.11e-04	2.25e-03	**1.37e-16**	**4.78e-17**	1.35e-02	1.17e-02	4.90e-02	5.51e-02
Ackley		**0**	**0**	3.80e-12	3.66e-12	9.08e-14	6.79e-14	1.26e-04	2.39e-04
Rastrigin		2.81e+01	9.37e+00	3.50e+01	6.95e+00	**2.98e-01**	**5.32e-01**	3.12e+00	1.65e+00

5.5 Conclusion

This chapter surveys the different parallel EDAs proposed in the literature relying on either exchanging individuals or probabilistic models.

Table 5.9: Results of all the algorithms for the CEC05 benchmark functions

Benc. Func.	Dim.	Adaptive_*gbest*		DMS-L-PSO		CPSO_S		TRIBES-D	
		Mean	**Std.**	**Mean**	**Std.**	**Mean**	**Std.**	**Mean**	**Std.**
f6	10	1.53e-01	1.91e-02	**5.00e-6**	**1.92e-05**	2.45e+01	2.94e+01	13.53e+00	21.53e+00
f9		2.36e+00	1.15e+00	**0**	**0**	**0**	**0**	**0**	**0**
f10		**2.95e+00**	**1.58e+00**	4.48e+00	1.27e+00	3.82e+01	1.89e+01	9.65e+00	3.43e+00
f11		**3.31e+00**	**1.96e+00**	4.76e+00	6.99e-01	6.67e+00	1.73e+00	4.06e+00	1.08e+00
f12		1.10e+03	1.23e+03	**2.48e+00**	**4.36e+00**	4.39e+02	6.69e+02	1.15e+03	6.68e+02
f13		6.34e-01	2.01e-01	3.77e-01	9.26e-02	**2.75e-01**	**1.46e-01**	4.27e-01	1.08e-01
f14		**2.42e+00**	**3.89e-01**	2.66e+00	2.44e-01	3.80e+00	3.53e-01	2.89e+00	5.09e-01
f6	30	**2.77e+01**	**2.78e+01**	7.53e+01	5.63e+01	8.54e+01	7.57e+01	5.87e+01	5.06e+01
f9		2.30e+01	5.41e+00	2.28e+01	5.30e+00	**3.32e-02**	**1.82e-01**	3.06e+00	1.73e+00
f10		**2.92e+01**	**1.94e+01**	4.46e+01	9.34e+00	1.78e+02	4.35e+01	1.45e+02	3.21e+01
f11		2.74e+01	6.56e+00	3.13e+01	8.98e-01	**2.11e+01**	**3.62e+00**	2.96e+01	2.26e+00
f12		1.50e+05	1.73e+05	**9.17e+02**	**1.34e+03**	6.60e+03	4.83e+03	1.05e+05	3.85e+04
f13		3.10e+00	7.38e-01	3.05e+00	5.25e-01	**1.15e+00**	**2.25e-01**	2.53e+00	7.03e-01
f14		1.26e+01	4.64e-01	**1.23e+01**	**3.36e-01**	1.32e+01	4.97e-01	1.29e+01	2.95e-01
f6	50	5.58e+01	3.08e+01	**1.31e+00**	**1.67e+00**	1.37e+02	1.52e+02	9.23e+01	8.24e+01
f9		5.48e+01	8.45e+00	5.98e+01	1.15e+01	**6.63e-02**	**2.52e-01**	5.47e+00	2.76e+00
f10		**6.20e+01**	**2.61e+01**	9.72e+01	1.39e+01	3.43e+02	7.62+01	3.65e+02	7.50e+01
f11		4.92e+01	9.00e+00	6.00e+01	1.31e+00	**3.81e+01**	**5.41e+00**	5.39e+01	4.55e+00
f12		7.33e+04	6.12e+04	**3.53e+03**	**3.86e+03**	1.40e+04	1.31e+03	4.08e+05	1.12e+05
f13		5.52e+00	1.27e+00	6.03e+00	9.83e-01	**2.22e+00**	**4.51e-01**	4.38e+00	1.07e+00
f14		2.20e+01	4.09e-01	**2.13e+01**	**4.42e-01**	2.28e+01	5.05e-01	2.25e+01	2.98e-01

Table 5.10: Comparison of all the algorithms

Algorithm	**Classical Functions**		**CEC05 Benchmark Functions**		**Total Number of Cases**
	No. of Cases	**Best in**	**No. of Cases**	**Best in**	
Adaptive_*gbest*	7	Spherical,Ackley	6	f10	13
DMS-L-PSO	5	-	8	f12	13
CPSO_S	4	Rastrigin	8	f9,f13	12
TRIBES-D	5	Spherical	1	-	6

The chapter proposes a new heterogeneous cooperative PSO/EDA algorithm based on the exchange of probability models. The model is considered as a heterogeneous approach because the cooperating PSO/EDA algorithms used different methods to sample the search space.

The model utilizes two different algorithms, namely, EDA-PSO and PSO_Bounds. The two algorithms exchanges their probability models every pre-determined number of iterations. Each algorithm converts the received model into an equivalent model that is in the same form of its resident one. The PSO_Bounds algorithm combines the received model with its resident one and continues with the search. On the other hand, EDA-PSO uses the combined received-resident model to generate new particles replacing its worst particles.

The new cooperative model produces better results than its individual components for different problem sizes. Studying the convergence behavior of the cooperative model, it is shown that the model has a behavior that is similar to the component performing better in the search, even if this component changed during the search process. The cooperative model does not produce good results for a small number of situations where one of its components had a very poor performance in the function under study.

Migrating a probability model is compared to the classic migration of particles. It is shown that there's no migration scheme outperforms the other on all the benchmark function.

Finally, a simple adaptive model is proposed. In this model, the flow of information is adaptively controlled in a sliding window approach based on its components performance during the search. The adaptive version significantly improves the results over the non-adaptive version and is able to increase the number of cases in which it outperforms its components. The adaptive_*gbest* version is also shown to be very competitive with some state-of-the-art cooperative PSO algorithms when applied to the benchmark functions under study.

Chapter 6

Particle Swarm Optimization for FPGA Placement

The placement problem in Field Programmable Gate Arrays (FPGAs) is crucial to achieve the best performance. Simulated annealing has been the main optimization algorithm used to solve it. In this chapter, two different PSO versions are applied to the FPGA placement problem in order to find the optimum logic blocks and IO pins locations in order to minimize the total wire-length. One version solves the the problem entirely in the discrete search space while the other version solves it in the continuous domain. Different cooperative models of both versions are also investigated. All the algorithms are implemented and applied to several well-known FPGA benchmarks with increasing dimensionality. Finally, both EDA-PSO and PSO_Bounds are applied to the problem as well as the cooperative model with probability models migration.

6.1 FPGAs Placement Problem

FPGAs are digital circuits that provide a programmable alternative to Application Specific Integrated Circuits (ASIC) designs for prototyping and small-volume production. The market shares of FPGAs had witnessed a huge increase recently since FPGA vendors started providing a variety of FPGA sizes for different applications. The design process of FPGAs involves synthesizing a user defined circuit and placing it on the programmable resources of FPGAs. The placement problem in FPGAs has always been the limiting factor for FPGA performance. The FPGA placement problem is a combinatorial problem where the logic and IO blocks are distributed among the available physical locations to either minimize the total wire-length or minimize the delay along the

critical path. The most widely used optimization algorithm in the FPGA placement problem is Simulated Annealing (SA) [75].

FPGAs consist of programmable logic resources (logic clusters) embedded in a sea of programmable interconnects (routing channels), as shown in Figure 6.1. Moreover, programmable IO pads are distributed along the edges of the fabric (Figure 6.1).

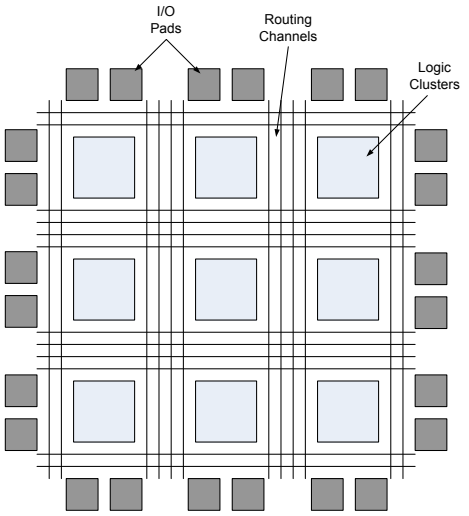


Figure 6.1: FPGA layout.

The programmable logic resources are configured to implement any logic function, while the interconnects provide the flexibility to connect any signal in the design to any logic resource. FPGA logic blocks are made of programmable Look-Up Tables (LUTs). Logic blocks and the programmable interconnects are controlled by built-in configuration SRAM cells, which control their operation. The process of programming the FPGA employs transforming the design to a series of zeros and ones, which are transferred to the configuration SRAM cells and are used to configure the programmable logic and interconnects.

A typical CAD flow for FPGAs is shown in Figure 6.2. Designs are synthesized by mapping them to the LUT FPGA architecture. Afterwards, the synthesized LUTs are grouped into a group of logic clusters that correspond to that of the physical FPGA structure. In the placement design phase of FPGAs, the logic blocks and the IO blocks of the given design are distributed among the physical logic blocks and IO pads, respectively, in the FPGA fabric. The connections between the placed logic clusters are established by programming the routing resources during the routing phase.

Placement algorithms try to minimize the longest delay along the paths in the circuit and/or the total wire length. In this work, the cost function is chosen to be the total wire length of the

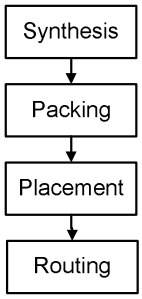


Figure 6.2: FPGA design CAD flow.

design. The wire length is calculated as the length of the bounding box of each wire. For example, if block i is placed in location (x_i, y_i) and block j is placed in location (x_j, y_j), then the length of the wire connecting them is calculated as

$$\text{wirelength} = |x_i - x_j| + |y_i - y_j|. \tag{6.1}$$

Simulated-annealing placement algorithms are used by the VPR tool [75]. The logic blocks are randomly placed in the FPGA fabric. Afterwards, random blocks are selected as candidates for swapping, a random location is selected and the swap is performed. Swaps that reduce the quality of the cost function are allowed initially, but the probability of accepting such swaps decreases with the number of iterations.

In [76], a comparison between the quality of a number of different placement algorithms is performed. It was found that the VPR-based algorithm results in the best placement quality in terms of the placement cost function. As a result, in this work, the results obtained by the PSO placement algorithms will be compared to those achieved by the VPR tool.

6.2 Discrete PSO

Different discrete versions have been proposed for PSO in the literature [54, 77–88]. In [54], a binary discrete PSO version was proposed where particle trajectories are represented as changes in the probability that a coordinate will be either '1' or '0', and applied to the five functions of the De Jong testbed. A discrete PSO was used to solve the task assignment problem in [77], which achieved better results than a standard genetic algorithm. Another discrete PSO version was also proposed in [78].

In [79,80], the authors solved the TSP problem using two similar discrete PSO algorithms. Both

algorithms located the optimal solutions for the studied problems. However, the TSP instances solved were of low dimensionality, 14 and 17, respectively.

The authors of [81] proposed a discrete PSO algorithm for solving the generalized TSP (GTSP) problem. The proposed algorithm had a modified subtraction operator and employed two local searching techniques for accelerating the speed of convergence and was applied to problems with up to 40 groups.

In [82] a variant PSO (VPSO) was applied to the steelmaking charge plan problem, where the authors used a discrete presentation. The charge plan scheduling problem was modeled as a TSP problem. The particles position was modeled as an array, where the array index is the slab number and the array element is the charge number. The velocity and position update equations were implemented as a series of crossover and mutation operators. A group of particles were updated by subtracting (instead of adding) the velocity in order to change the search direction. The method was successfully applied to problems with dimensionalities up to 30.

A modified PSO (MPSO) algorithm for solving the TSP problem was proposed in [83]. In MPSO , each particle moves towards its personal best and either *gbest* or a randomly selected particle. At the beginning, there is a higher probability that *gbest* is not followed. Towards the end of the search, this probability is decreased and each particle follows the *gbest* instead. Velocity mutation was used if the best found solution does not change for a specified number of iterations. The velocity vector was composed of a sequence of adjustment operators. Each adjustment operators $V(i, j)$ was applied by removing the node at index i, inserting it at index j and shifting the rest of the position vector. The method was only applied to two instances of a dimensionality of 14 and 29 performing a large number of function evaluations, 2×10^5 and 3×10^5, respectively.

A sequential PSO (SPSO) was used for the graphic presentation of Group Method of Data Handling (GMDH) networks in [84]. The objective was to find the optimum sequence of nodes which will reduce the number of intersections in the graph for better visualization. The particle position was a vector containing the nodes to be sorted. The method was applied to a graph consisting of 20 nodes. However, no details were given about the way the particle position and velocity were updated and the results were not compared to any other method. The minimum number function evaluations required to achieve good results was found to be 3×10^4.

In [85], a discrete PSO was applied to the weapon-target assignment (WTA) problem. Each particle position is represented as a vector, where the index is the target and the value is the weapon assigned to it. The velocity was also a vector found by the permutation of two position

vectors. The method was successfully applied to problems up to a dimensionality of 60. However, the number of function evaluations used in the experiments was not reported.

A dual similar PSO algorithm (DSPSOA) was proposed in [86] for solving the job-shop scheduling (JSS) problem. The authors used the crossover operator between two positions in order to come up with a velocity vector. The crossover operator was also applied between the current particles position and velocity to result in the new position. In between, velocity and position vectors are mutated. The algorithm consisted of two PSO algorithms, referred to as the outer and inner PSOs. Both algorithms used different operators for the crossover and mutation. The algorithm was only applied to a small-sized problem consisting of 10 jobs and 5 machines with 600 function evaluations for the outer PSO and 3000 function evaluations for the inner one.

For a single machine job scheduling problem, a discrete PSO algorithm was proposed in [87]. The particle position was made of a vector of $n+b$ jobs, where n is the number of jobs and b is the number of dummy jobs. To update the position, a swap operator is applied to the position with a certain probability w. Then, a one-cut crossover is applied between the swapped position and the personal best with a probability c_1 to obtain a more updated position. Finally, a two-cut crossover operator is performed between the updated position and the global best with a probability c_2 to find the new current position. Local search was applied to the *gbest* at every iteration to improve the solution quality. Their method was applied successfully to problems having up to 1000 jobs. The function evaluations performed had a maximum of 1.6×10^4 for the smallest problem and 3×10^5 for the biggest one.

PSO was also applied to the Field Programmable Gate Arrays (FPGA) placement problem in [88]. The authors solved the problem in the continuous search space. Moreover, the algorithm was only applied to two problems without comparing the results to any other optimization approach, and the best known solution for these problems were not reported.

In [89] , the authors proposed a discrete PSO algorithm, referred to as SetPSO, to optimize the structure of RNA molecules. The addition of two particles was defined as the union of the two sets. On the other hand, the subtraction was defined as the set-theoretic difference of the two sets. SetPSO was applied to 4 benchmarks of dimensionality 118, 122, 784 and 945 performing 35000 function evaluations. The percentage of the correct pairs detected were 89.%, 76.3%, 36.7%, and 15.9%, respectively.

6.3 Discrete PSO Placement Algorithm

In the Discrete PSO (DPSO) version used in this work, which is based on [79], each particle position corresponds to the available physical locations in the whole FPGA array. For example, if the FPGA consists of a 10×10 array, the particle position is an array of 100 elements, where each element represents one location in the FPGA array. Hence, the element index is the location number on the FPGA array and its value is the block number occupying that location. Elements denoting empty locations have the value of -1. Figure 6.3 shows an example for the DPSO problem formulation for a 3x3 FPGA array.

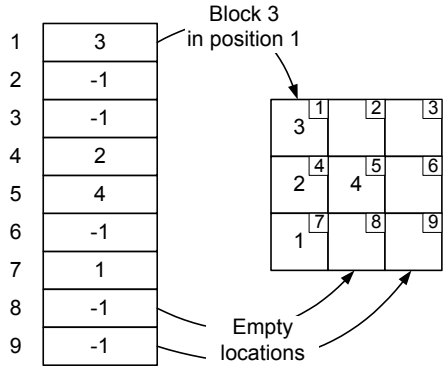


Figure 6.3: DPSO formulation.

In the used DPSO formulation, the particle velocity is represented as a sequence of swaps. For example, a velocity $v = \{(1, 4)\}$ represents a swap between the first and fourth elements of the particle position, *i.e.*, block 3 is swapped with block 2 in Figure 6.3. Since v contains only one swap operation, then the size of v is equal to 1. The two blocks swapped should be of the same type, either logic blocks or IO pins, swapping an IO pin with a logic block is not allowed. During particle initialization, the velocities for each particle are initialized with random velocities of random size ranging from 0 to V_{max}. The value of V_{max} is selected by conducting several experiments with different values for V_{max} and examining its impact on the value of the cost function.

6.3.1 DPSO Operations

From the conventional PSO relations given in Chapter 2, the position update procedure consists of three different operations; addition of a velocity to a position, subtracting two positions, and multiplying a velocity by a constant.

The addition of a velocity v to a position x is carried out by applying the sequence of swaps defined by v to x. This results in another position, which has the same size as the original position x. The swaps are applied in the same order as they are listed in v. An example for the addition operator is depicted in Figure 6.4.

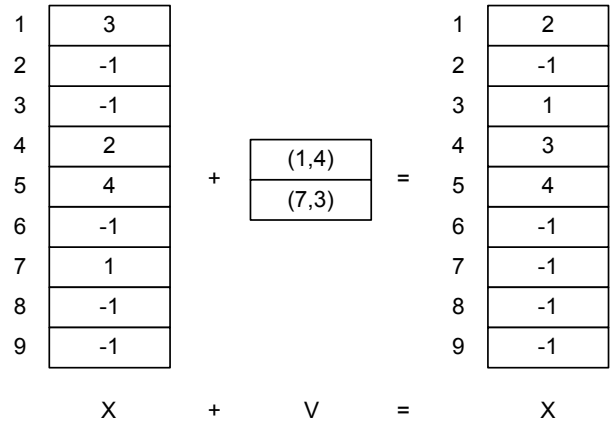


Figure 6.4: A position plus velocity example.

Subtracting a position vector x_1 from another position x_2 is performed by identifying the sequence of swaps v that would transform x_1 to x_2. The maximum complexity of this operation is proportional to the position size, which can grow up significantly depending on the problem size. Hence, the velocity size is clamped to V_{max}, to reduce the computational complexity of the algorithm. As a result, applying the swaps given in v might not result in transforming x_1 to x_2 exactly, if the distance between them is larger than V_{max}.

Multiplying a velocity v by a constant c affects the velocity size. If c is equal to zero, the size of v is set to zero. If c is less than 1, v is truncated by removing swaps from the end of the vector such that the size of the resulting velocity is equal to the size of the original v multiplied by c. If c is larger than one, new swaps are added to the end of v to increase its size by c. The newly added swaps are extracted from the top of v. Figure 6.5 shows an example for the multiplication of a velocity by a constant.

6.3.2 DPSO Problem Formulation

In the proposed DPSO problem formulation, each position element corresponds to a unique physical location on the FPGA, including both logic blocks and IO pins, as explained above. To make sure that no swaps occur between logic blocks and IO pins, the position vector x of each particle is divided into two vectors x_{Logic} and x_{IO}. Moreover, the velocity vector v of the particle is also

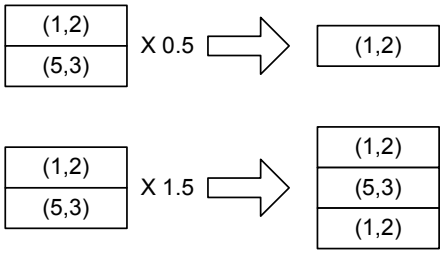


Figure 6.5: A constant times velocity example.

composed of two parts v_{Logic} and v_{IO}. To update the position and the velocity of each particle, equations (2.1) and (2.2) are applied twice, using the newly defined operators in Section 6.3.1, for both the logic and IO parts. However, before updating the position, the velocities are mutated by applying one random swap to both velocity parts as a means for overcoming premature convergence. In order to evaluate the fitness of the new particle position, both parts are used to construct the complete position x. This implementation ensures that the dependency between the logic blocks and IO pins (due to connections between them) is kept intact. The algorithm used for DPSO in this work is shown in Algorithm 6.1.

6.3.3 Local Minima Avoidance

In [79], the author proposed several approaches to enhance local search ability and avoid premature convergence for discrete PSO algorithms. If the solution does not improve for a specified number of iterations, the different particles move back to their *Pbest* positions and perform a *lazy descent* type of search in order to find a better solution for a limited number of iterations. After that, if the swarm is too small (due to having particles sharing the same position) the swarm is completed with a newly initialized set of particles. More local search strategies were incorporated as well such as *deep descent* and *adaptive descent*.

In our experiments, only *local descent* is used as shown in Algorithm 6.2, as other methods increased the time complexity of the algorithm without resulting in an improvement in the solution quality. The lazy descent method employed in this work is invoked if the *gbest* does not improve for a specific number of iterations. It was found that a threshold of 3 non-improving iterations works best for a wide range of the benchmarks tested. The lazy descent generates a random velocity v_{lazy} of size one, *i.e.*, only one swap. v_{lazy} is then added to the particle *pbest*. If the particles best solution is improved, the lazy descent terminates and *pbest* is updated, as well as *gbest*, if needed. Otherwise, the above steps are repeated for a specific number of iterations. If no improved position

Algorithm 6.1 DPSO implementation.

Require: Max_Function_Evaluations

1: Initialize the swarm
2: Max_Iterations = $\frac{Max\_Function\_Evaluations}{Num\_Particles}$
3: iter_number = 1
4: **while** $iter\_number \leq Max\_Iterations$ **do**
5: **for** each particle i **do**
6: Update V^i_{IO}, V^i_{Logic}
7: Mutate V^i_{IO}, V^i_{Logic}
8: Update X^i_{IO}, X^i_{Logic}
9: Construct $X^i = [X^i_{\text{IO}}, X^i_{\text{Logic}}]$
10: Update $pbest^i$
11: **end for**
12: Update *gbest*
13: **if** *gbest* is not improved **then**
14: Local Minima Avoidance()
15: Re-initialize velocities
16: **if** $iter\_number \geq \frac{Max\_Iterations}{2}$ **then**
17: Scatter Particles()
18: **end if**
19: **end if**
20: iter_number = iter_number + 1
21: **end while**
22: **return** *gbest*

is found, the lazy descent terminates without changing the original *pbest* position.

In addition to lazy descent, the velocities of all the particles are re-initialized to explore new areas of the search space. The work also employs a *scattering* process to jump over local minima. Similar to the lazy descent, if the *gbest* does not change for a specific number of iterations, the particles positions, not their *pbests* are re-initialized. Instead of scattering (re-initializing) a big number of particles, only the particles that are too close to the *gbest* are scattered. This strategy has a low computational cost while not compromising the provided solution quality. Moreover, the scattering algorithm is only invoked after the number of iterations performed exceeds half the total number of iterations. It was noticed that as the DPSO algorithm progresses, the particles

positions tend to group around the *gbest*, thus they are only scattered towards the end of the algorithm.

Algorithm 6.2 The lazy descent method for the discrete algorithm.

```
1: while iter_number ≤ 5 do
2:    for each particle i do
3:       Temp = pbest^i
4:       Generate V_lazy
5:       Temp = Temp + V_lazy
6:       if f(Temp) < f(pbest^i) then
7:          X^i = Temp
8:          pbest^i = Temp
9:          Break
10:      end if
11:   end for
12:   iter_number = iter_number + 1
13: end while
```

6.4 Discrete Cooperative PSO

The discrete cooperative version investigated is based on decomposing the search space into two sub-spaces, the Logic sub-space and the I/O sub-space. Two discrete PSO swarms are employed, each optimizing a different sub-space. The overall solution vector is constructed using the *gbest* of each swarm. To update the fitness value for a certain particle i in a swarm, a solution vector is used with that particle and the *gbest* the other swarm. This approach was originally proposed for continuous non-linear function optimization in [9] and was referred to as cooperative PSO (CPSO), which is illustrated in Figure 6.6.

In FPGA placement, it is not quite clear if splitting the problem search space into the Logic and I/O spaces would improve the solution since this is closely related to the degree of dependency between the two sub-spaces. The main advantage though, is that adopting this model will reduce the computational cost required by the original algorithm since all the mathematical operations would be performed using small vectors (sub-spaces) rather than a big vector (the overall solution). Algorithm 6.3 shows the steps taken to apply DCPSO in this work.

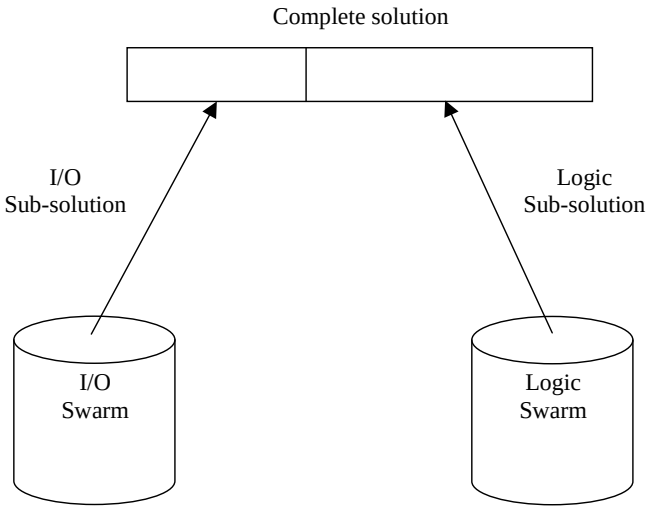


Figure 6.6: The discrete cooperative PSO.

Algorithm 6.3 Proposed DCPSO implementation.

Require: Max_Function_Evaluations

1: Initialize the I/O swarm
2: Initialize the Logic swarm
3: Max_Iterations = $\frac{Max\_Function\_Evaluations}{Num\_Particles}$
4: iter_number = 1
5: **while** $iter\_number \leq Max\_Iterations$ **do**
6: Update I/O swarm
7: Update the complete solution
8: Update Logic swarm
9: Update the complete solution
10: iter_number = iter_number + 1
11: **end while**
12: **return** The complete solution

6.5 Continuous PSO Placement Algorithm

In the continuous PSO version proposed in this work, each particle element corresponds to an IO or logical block to be placed in the available physical locations. For example, if there are 9 IO blocks and 15 Logic blocks to be placed, the particle position is an array of 24 elements, where each element represents the location number in which this block should be placed. Hence, the element index is the block number to be placed and its value is the location number it will be placed in.

The particle velocity is represented as a vector of real numbers. During particle initialization, the velocity for each particle is initialized with random values from 0 to V_{max}. The value of V_{max} is selected according to the domain size.

Unlike the discrete version, the continuous nature of the particles would sometimes cause their positions to have an invalid solution, which dictates the use of some repair mechanism. The method used for this is to check the particle element-by-element, if the location indicated by the value inside an element is already occupied by another block, a search is started for the closest empty location (to the left and right of the occupied one) in order to use it to place the block in hand.

6.5.1 Local Minima Avoidance

In our experiments, the *local descent* method is the only one used as was done in the discrete version. The lazy descent method, shown in Algorithm 6.4, is again invoked if the *gbest* does not improve for a 3 iterations. In the continuous version, the lazy descent only works with the *gbest*, it applies a sequence of local movements until the solution improves or a specific number of iterations has been reached.

As shown in Algorithm 6.4. There are two different types of local movements. One movement is to swap two randomly selected *IO* or *Logic* elements. While another movement is to mutate a randomly selected *IO* or *Logic* element. The mutation is done by re-initializing the chosen element to a random number uniformly distributed in the domain.

6.6 Results and Discussions

6.6.1 Experimental Setup

Both the discrete and continuous PSO algorithms are implemented and applied to several standard FPGA benchmark circuits of increased dimensionality. Table 6.1 lists the benchmark circuits used and their associated problem sizes, where $P(L, I)$ represents a problem with a total size of P having L Logic locations and I IO locations.

In this work, the performance of algorithms in minimizing the cost function is compared to that of the VPR tool. VPR employs an adaptive SA algorithm, where the number of iterations depends on the problem size. In this work, VPR is left to run until it reports its best found

Algorithm 6.4 The lazy descent method for the continuous algorithm.

```
1: while iter_number ≤ numberofparticles do
2:    Temp = gbest
3:    Generate a random number R
4:    if R < 0.25 then
5:       Swap two IO elements
6:    else if R < 0.5 then
7:       Swap two Logic elements
8:    else if R < 0.75 then
9:       Mutate an IO element
10:   else
11:      Mutate a Logic element
12:   end if
13:   if f(Temp) < f(gbest) then
14:      gbest = Temp
15:      Break
16:   end if
17:   iter_number = iter_number + 1
18: end while
```

solution. In order to have a fair comparison, the PSO algorithms are run to perform the same number of function evaluations performed by VPR.

In the DPSO algorithm, both c_1 and c_2 are set to 2, w is equal to 0.5, similar to [79] and V_{max} is set to 50. The lazy descent is applied if the *gbest* does not improve for 3 iterations and a maximum of 5 steps are performed every time. The results reported are the averages taken over 10 runs. In the continuous version, the same parameter values used in the previous chapters are adopted.

Table 6.2 shows the number of function evaluations performed by the PSO algorithms for each benchmark. After extensive experiments, it is found that the swarm size should be increased with the problem dimensionality.

Table 6.1: Problem sizes for the different benchmarks used.

Benchmark	Problem Size
cm42a	36(4, 32)
lion	57(9, 48)
b02	57(9, 48)
daio	57(9, 48)
dk27	80(16, 64)
b01	80(16, 64)
my_adder	80(16, 64)
count	80(16, 64)
s208.1	132(36, 96)
b9	132(36, 96)
s832	225 (81, 144)
s967	260 (100, 160)
ex5p	561(289, 271)
apex4	665(361, 304)

6.6.2 Experimental Results

The results of applying VPR and the PSO algorithms on several FPGA benchmarks are shown in Table 6.3. It can be seen that PSO produces better results than the VPR placement tool in problems with a dimensionality up to 60. When the dimensionality increases to 80, PSO produces results that are within a 5% margin from the results supplied by VPR. For larger-sized problems, the PSO is within 10% of VPR. Another observation from Table 6.3 is that the standard variation of PSO is less than that for the VPR for the small and medium sized benchmarks. This renders the PSO algorithm as being more reliable than VPR which is based on SA.

Table 6.4 and Table 6.5 show the results of applying the DCPSO and CPSO algorithms, which are both based on search space decomposition, as well as the average computational cost in seconds. The results show that the cooperative versions can maintain the same quality of solutions produced by the single swarm while minimizing the computational cost of the algorithm. In the discrete version, the computational cost is minimized by at least 60%. On the other hand, in the continuous version, the computational cost is minimized by at least 20%.

Figure 6.7, Figure 6.8, and Figure 6.9 plot the convergence behavior of VPR, DPSO and

Table 6.2: Swarm size and performed function evaluations.

Benchmark	**Swarm Size**	**Evaluations**
cm42a	40	14378
lion	60	5586
b02	60	5257
daio	60	5400
dk27	80	6832
b01	80	13944
my_adder	80	99016
count	80	98210
s208.1	80	31000
b9	80	100000
s832	80	135600
s967	80	204900
ex5p	100	825461
apex4	100	815423

Table 6.3: VPR, DPSO and continuous PSO Results for several FPGA benchmarks.

Benchmark	**VPR**		**DPSO**		**Continuous PSO**		**Error**
	Mean	**Std.**	**Mean**	**Std.**	**Mean**	**Std.**	**Margin**
cm42a	0.4286	0.0067	**0.4256**	0	**0.4256**	0	-0.7
lion	0.6088	0.0082	**0.6053**	0.0056	0.6073	0.0047	-0.57
b02	0.5649	0.0060	**0.5603**	0	0.5637	0.0056	-0.81
daio	0.6120	0.0175	0.61	0	**0.6040**	0.0097	-1.31
dk27	0.9659	0.0161	**0.9622**	0.0006	0.9704	0.0097	-0.38
b01	**1.2461**	0.0076	1.2509	0.0041	1.2677	0.0037	0.39
my_adder	**2.091**	0.0373	2.168	0.0426	2.2805	0.0460	3.68
count	**2.1592**	0.0420	2.2581	0.0394	2.24	0.0757	4.58
s208.1	**2.973**	0.0242	3.0298	0.0359	3.0846	0.0591	1.92
b9	**4.4795**	0.0325	4.7393	0.1150	4.7640	0.0450	5.80
s832	**11.7551**	0.0783	12.6215	0.2216	12.9677	0.1575	7.37
s967	**18.6722**	0.0869	20.3944	0.2994	20.3659	0.3322	9.22
ex5p	**103.1914**	0.3329	112.5834	1.8520	113.3372	0.7601	9.1
apex4	**117.049**	0.4911	127.0045	1.4052	131.3854	2.1338	9.3

Table 6.4: DCPSO results for several FPGA benchmarks.

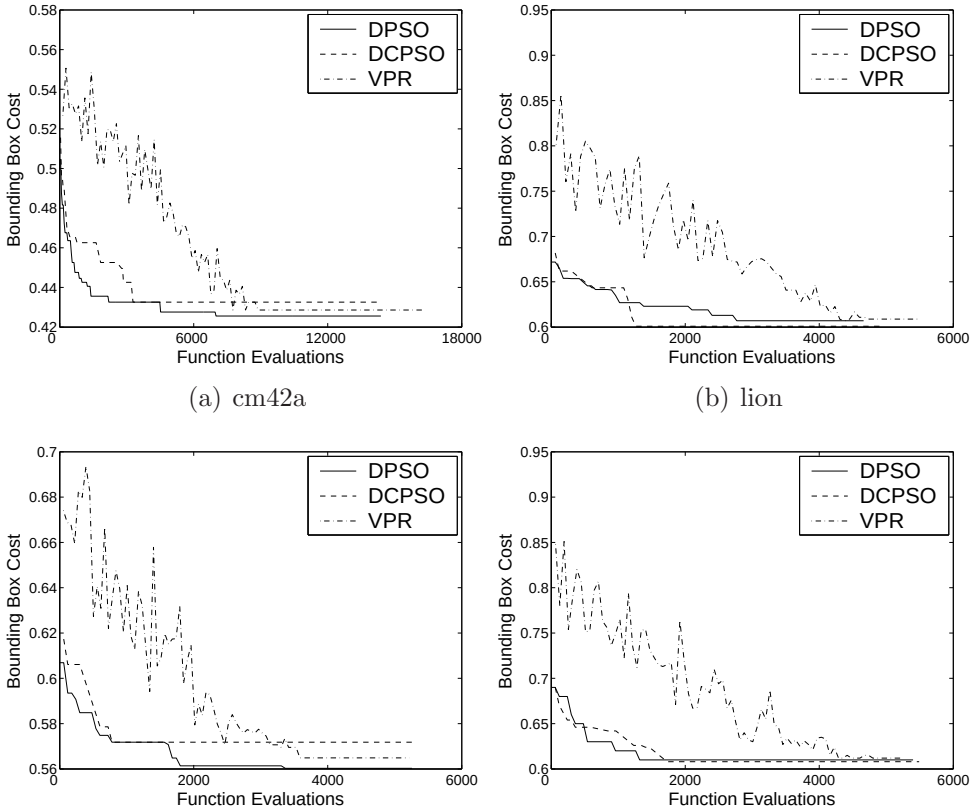
Benchmark	**DCPSO**		**Time**	**Time**
	Mean	**Std.**	**DCPSO**	**DPSO**
cm42a	0.4326	0.0048	0.049	0.202
lion	0.6010	0	0.020	0.063
b02	0.5695	0.0049	0.013	0.054
daio	0.6080	0.0042	0.020	0.078
dk27	0.9547	0.0154	0.026	0.097
b01	1.2996	0.0350	0.047	0.150
my_adder	2.2080	0.0484	0.577	1.973
count	2.2678	0.0430	0.073	4.580
s208.1	3.1973	0.0937	0.284	1.920
b9	4.8219	0.1128	1.493	4.759
s832	12.8385	0.2668	3.301	9.473
s967	20.3319	0.3960	5.948	17.377
ex5p	111.7615	0.9814	86.618	254.031
apex4	132.8206	8.4403	96.347	248.031

Table 6.5: CPSO results for several FPGA benchmarks.

Benchmark	**CPSO**		**Time**	**Time**
	Mean	**Std.**	**CPSO**	**Cont. PSO**
cm42a	0.4446	0.0032	0.041	0.051
lion	0.6318	0	0.009	0.011
b02	0.5614	0.0007	0.006	0.008
daio	0.6150	0.0053	0.010	0.014
dk27	0.9718	0	0.020	0.020
b01	1.3069	0.0253	0.031	0.042
my_adder	2.2590	0.0840	0.725	1.173
count	2.2865	0.0431	0.915	0.938
s208.1	3.1741	0.1464	0.266	0.337
b9	4.7753	0.1052	1.857	2.351
s832	12.6188	0.2768	3.896	4.707
s967	20.3744	0.3165	7.555	9.303
ex5p	113.5838	1.0051	108.8	125.167
apex4	131.5267	1.5720	116.359	134.777

DCPSO for all the benchmarks tested. It can be seen that both DPSO and DCPSO have very similar behaviors in most of the benchmarks. The DPSO and DCPSO algorithms always start

from a better quality solution due to the initialization of a population of candidate solutions. Both of them quickly identify good regions in the search space and move to these regions much faster than VPR. However, these algorithms fail to fine tune the final solution using local search. On the other hand, VPR manages to fine tune the solution better than DPSO and DCPSO because the SA algorithm is superior in local search.



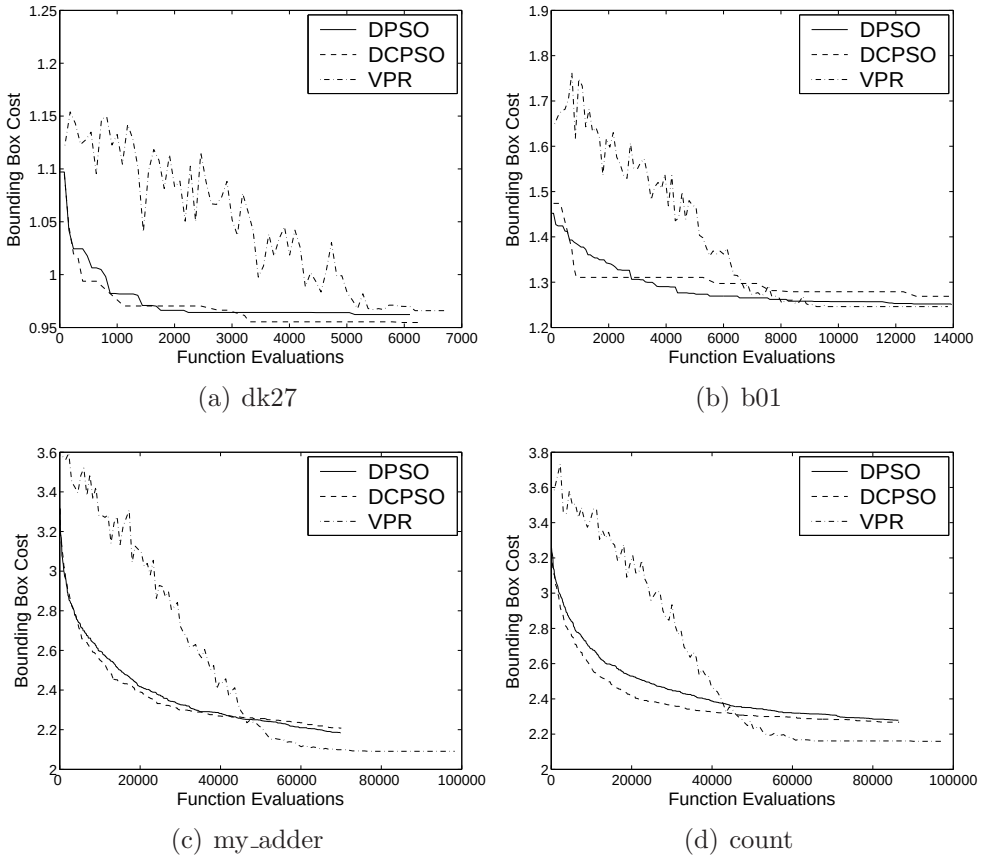
(a) cm42a (b) lion

(c) b02 (d) daio

Figure 6.7: Convergence behavior for problems with a dimensionality within 60.

Another cooperative continuous model is applied for this problem, which is a 2-cooperating swarms approach similar to the one experimented with in Chapter 3. The model is tested using different synchronization periods. Table 6.6 shows the best result achieved by the model for every benchmark and the synchronization period at which it was obtained. The results show that adopting such a cooperative system is generally better than using a single swarm as was concluded in Chapter 3.

EDA-PSO and PSO_Bounds were also applied to solve this problem. The results shown in Table 6.7 illustrate that PSO_Bounds has the better performance as it produces the best results on most of the benchmarks.



(a) dk27 (b) b01

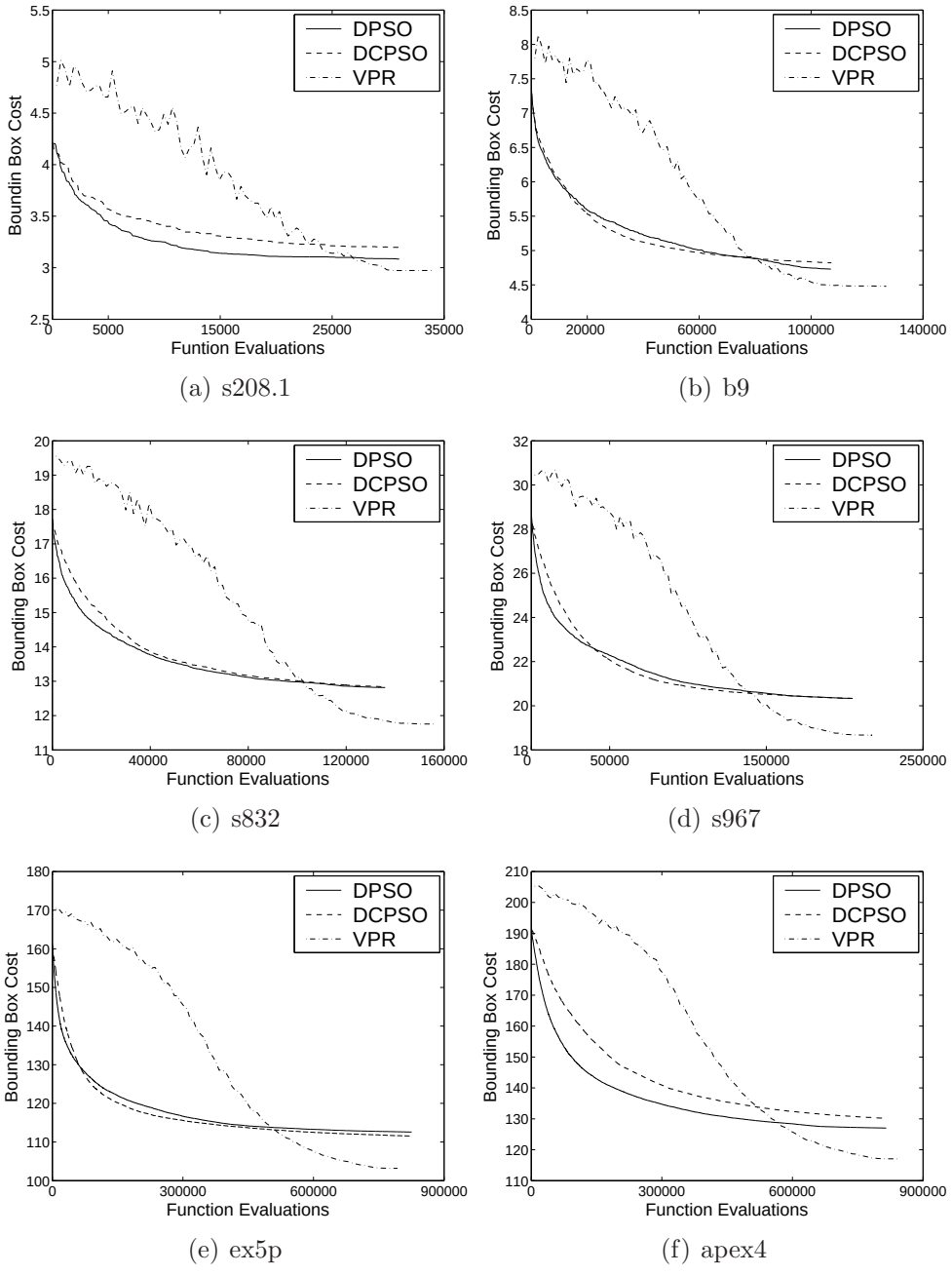
(c) my_adder (d) count

Figure 6.8: Convergence behavior for problems with a dimensionality within 80.

The adaptive model using probability models migration is also applied and the results are shown in Table 6.8. The results begin to deteriorate for larger benchmarks, this could be due to the repair mechanism adopted for the particles positions. The exchanged information is either on the form of specific bounds or the mean and standard deviation of a Gaussian distribution. Although the PSO equations take this information into account when updating the position, this information could be lost by the repair mechanism. The repair mechanism could replace the position of a certain block to the nearest empty location, which might be outside the specified bounds or not covered by the Gaussian distribution.

6.7 Conclusions

This chapter introduces two different versions of the PSO algorithm to be applied to the FPGA placement problem. The algorithms are tested on several FPGA benchmarks with increased dimensionality and compared to the academic VPR placement tool, which is based on adaptive



(a) s208.1 (b) b9

(c) s832 (d) s967

(e) ex5p (f) apex4

Figure 6.9: Convergence behavior for problems with a dimensionality above 100.

simulated annealing. The results show that PSO outperforms VPR (providing robust and better results with a faster convergence rate) for benchmarks with sizes within 60. In larger benchmarks, the error margin is within 5% for problems with sizes up to 80 and 10% for problem sizes up to 665.

The work also applies two different cooperative models. The use of a cooperative PSO model, which is based on space decomposition, leads to maintain the same quality of solutions produced

Table 6.6: Results of the continuous cooperative model.

Benchmark	**Synchronization Period**	**Mean**	**Std.**	**Significantly better**
cm42a	*10*	*0.4256*	*0*	*The same*
lion	**50**	**0.6018**	**0**	**Yes**
b02	**10**	**0.5603**	**0**	**Yes**
daio	*50*	*0.6010*	*0.0032*	*The same*
dk27	**25**	**0.9427**	**0**	**Yes**
b01	**50**	**1.2601**	**0.0146**	**Yes**
my_adder	**50**	**2.1820**	**0.0447**	**Yes**
count	*10*	*2.2655*	*0.0433*	*The same*
s208.1	*50*	*3.0622*	*0.0822*	*The same*
b9	*10*	*4.7760*	*0.0531*	*The same*
s832	**25**	**12.8623**	**0.1941**	**Yes**
s967	100	20.5773	0.3344	No
apex4	*10*	*131.6418*	*1.9808*	*The same*
ex5p	*10*	*113.6244*	*1.0926*	*The same*

Table 6.7: PSO, EDA_PSO and PSO_Bounds results for several FPGA benchmarks.

Benchmark	**PSO**		**EDA-PSO**		**PSO_Bounds**	
	Mean	**Std.**	**Mean**	**Std.**	**Mean**	**Std.**
cm42a	**0.4256**	0	**0.4256**	0	**0.4256**	0
lion	0.6073	0.0047	**0.6018**	**0**	**0.6018**	**0**
b02	**0.5637**	**0.0056**	0.5672	0.0060	**0.5661**	**0.0061**
daio	0.6040	0.0097	0.6140	0.0052	**0.6**	**0**
dk27	0.9704	0.0097	**0.9487**	**0.0126**	**0.9455**	**0.0044**
b01	1.2677	0.0037	1.2871	0.0136	**1.2470**	**0.0006**
my_adder	**2.2805**	**0.0460**	2.2985	0.0676	2.2938	0.0178
count	2.24	0.0757	**2.1830**	**0.0330**	**2.1950**	**0.0633**
s208.1	**3.0846**	**0.0591**	**3.0758**	**0.0355**	**3.0896**	**0.0475**
b9	**4.7640**	**0.0450**	4.8307	0.1369	**4.78**	**0.0987**
s832	12.9677	0.1575	**12.6060**	**0.1755**	12.9	0.1703
s967	**20.3659**	**0.3322**	**20.5019**	**0.31051**	**20.4259**	**0.3623**
ex5p	**113.3372**	**0.7601**	**112.9794**	**1.2631**	**113.5139**	**1.3960**
apex4	**131.3854**	**2.1338**	132.9890	1.5656	**132.2794**	**1.4404**

by a single swarm while minimizing the time requirement of the algorithm by at least 60% in the discrete domain and 20% in the continuous domain. The use of a 2-cooperating swarms model

Table 6.8: Results of the adaptive cooperative model based on probability models migration.

Benchmark	**Synchronization Period**	**Mean**	**Std.**	**Significantly better**
cm42a	*25*	*0.4256*	*0*	*The same*
lion	*10*	*0.6018*	*0*	*The same*
b02	**25**	**0.5603**	**0**	**Yes**
daio	**100**	**0.59**	**0**	**Yes**
dk27	**100**	**0.9427**	**0**	**Yes**
b01	25	1.2589	0	No
my_adder	*100*	*2.1740*	*0.0470*	*The same*
count	*100*	*2.2838*	*0.0762*	*The same*
s208.1	50	3.1089	0.0274	No
b9	*100*	*4.7588*	*0.1079*	*The same*
s832	100	12.8936	0.1786	No
s967	50	20.7097	0.3558	No
apex4	25	133.8797	1.6885	No
ex5p	50	114.5790	0.8244	No

proved to better than the single swarm version for most of the cases.

As for the PSO and EDA hybrids, PSO_Bounds shows to have better performance than EDA-PSO. However, adopting the cooperative model that is based on probability models migration suffers in the large-sized problems due to the influence of the repair mechanism.

Chapter 7

Conclusions and Future Work

7.1 Conclusions

This thesis presented a comprehensive survey of all the cooperative PSO models proposed in the literature, these model were categorized according to the application they were designed for. A classification scheme was proposed to classify the surveyed models based on the type of the decomposition adopted by the model and the approach used for placing the particles into the different swarms.

From this survey, similarities and differences of these models were identified based on four design key issues that need to be selected when implementing a cooperative model. These decisions gave answers to the four questions: Which information to share? When to share it? Whom to share it with? and What to do with it?

The work in the thesis continued by performing a complete experimental study on one of the cooperative PSO models proposed in the literature in order to understand how the performance changes under the influence of the design issues previously identified. The addressed issues include the exchange of the *gbest* information vs. the exchange of complete particles, changing the number of iterations between successive communication steps, changing the number of cooperating swarms, changing the neighborhood topology, changing the method for selection and replacement of exchanged particles, and changing the number of exchanged particles.

In order to investigate the idea of exchanging probability models instead of particles migration. A new PSO and EDA hybrid was proposed which was based on PBIL. The proposed method outperforms other proposed hybrids on the set of shifted and/or rotated multimodal function.

The work proposed a new heterogeneous cooperative PSO model, which is based on the exchange of probability models rather than the classical migration of particles. In order to accommodate the differences between the exchanged models, each swarm performed a *model conversion* step on the received probability model to transform it into a model similar to its own and a *model combination* step between the resident and received models before continuing with the search. The proposed model was shown to be competitive with other state-of-the-art cooperative PSO algorithms.

Finally, two versions of PSO, discrete and continuous, were applied to the FPGA placement problem. The approaches were compared to the academic tool VPR, which is based on adaptive simulated annealing. It was shown that both versions outperform VPR (providing robust and better results with a faster convergence rate) for benchmarks with sizes within 60. In larger benchmarks, the error margin is within 5% for problems with sizes up to 80 and 10% for problem sizes up to 665. The work applied a cooperative PSO version where the placement of the I/O and logic block is being optimized by different swarms. The use of this model maintained the same quality of solutions produced by a single swarm while minimizing the time requirement of the algorithm by at least 60% in the discrete domain and 20% in the continuous domain. The use of a 2-cooperating swarms model proved to better than the single swarm version for most of the cases. For the PSO and EDA hybrids, PSO_Bounds showed to have better performance than EDA-PSO. However, adopting the cooperative model that is based on probability models migration suffered in the large-sized problems due to the influence of the repair mechanism.

7.2 Future Work

Different future research direction could be followed from this point. One direction is to conduct a study similar to the one carried out in this thesis but with using an *asynchronous* communication approach. This is an important study as many of the best results reached by the cooperative model was obtained at long synchronization periods indicating that an *asynchronous* type of communication would be beneficial.

One disadvantage of PSO_Bounds algorithm proposed in this thesis is its poor performance in unimodal functions. The reason for this is due to the fast convergence of PSO leading the continuously updated bounds to overlap in some dimensions. When this happens, the movement in these dimensions will stop causing the algorithm to stagnate. One approach to overcome this problem is to reset the bounds to the initial domain when the width of the allowable domain of a

certain dimension drops under a predetermined threshold. One could also further re-initialize the particles current positions when this happens keeping their *pbests* as they are so as not to lose any useful information.

Another area for possible future research is to improve the performance of the proposed hybrid cooperative model by employing a better-informed information exchange method. This could be done by updating the *model combination* step by only replacing dimensions when the exchange would seem beneficial, instead of being completely random. Also, instead of having both components performing 50% of the function evaluations, the model could adaptively set this ratio according to each component's performance during the search.

For the FPGA placement problem, the performance of the PSO algorithms could be enhanced by incorporating additional approaches to escape local minima. This is expected to improve the results for large benchmarks by providing the algorithms with fine tuning ability.

Bibliography

[1] J. Kennedy and R. C. Eberhart, "Particle swarm optimization," in *Proc. of IEEE International Conference on Neural Networks*, vol. 4, 1995, pp. 1942–1948.

[2] R. C. Eberhart and J. Kennedy, "A new optimizer using particle swarm theory," in *Proc. of the 6th International Symposium on Micro Machine and Human Science*, 1995, pp. 39–43.

[3] T. G. Craninc, M. Toulouse, and M. Grendeau, "Synchronous tabu search parallelization strrategies for multicommodity location-allocation with balancing requirements," Centre de recherche sur les transports, Universite de Montreal, Tech. Rep. 934, 1993.

[4] T. G. Crainic and M. Grendeau, "Cooperative parallel tabu search for capacitated network design," *Journal of Heuristics*, vol. 8, pp. 601–627, 2002.

[5] M. Nowostawski and R. Poli, "Prallel genetic algorithms taxonomy," in *Proc. 3rd international Conference on Knowledge-Based Intelligent Information Engineering Systems*, 1999, pp. 88–92.

[6] E. Cantu-Paz, "A survey pf parallel genetic algorithms," The University of Illinois, Tech. Rep. IlliGAL 97003, 1997. [Online]. Available: ftp://ftp-illigal.ge.uiuc.edu/pub/papers/IlliGALs/97003.ps.Z

[7] M. Middendorf and F. Reischle, "Information exchange in multi colony ant algorithms," in *Proc. 3rd workshop on Biologically Inspired Solutions to Parallel Processing Problems*, 2000, pp. 645–652.

[8] M. Middendorf, F. Reischle, and H. Schmeck, "Multi colony ant algorithms," *Journal of Heuristics*, vol. 8, pp. 305–320, 2002.

[9] F. van den Bergh and A. P. Engelbrecht, "A cooperative approach to particle swarm optimization," *IEEE Transactions on Evolutionary Computation*, vol. 8, no. 3, pp. 225–239, 2004.

[10] S. Baskar and P. N. Suganthan, "A novel concurrent particle swarm optimization," in *Proc. IEEE Congress on Evolutionary Computation*, vol. 1, 2004, pp. 792–796.

[11] F. van den Bergh and A. P. Engelbrecht, "Effect of swarm size on cooperative particle swarm optimizaters," in *Proc. Genetic and Evolutionary Computation Conference*, 2001.

[12] M. A. Potter and K. A. de Jong, "A cooperative coevolutionary approach to function optimization," in *Proc. 3rd Parallel problem Solving from Nature*, 1994, pp. 249–257.

[13] T. Peram, K. Veeramachaneni, and C. K. Mohan, "Fitness-distance-ratio based particle swarm optimization," in *Proc. IEEE Swarm Intelligence Symposium*, 2003, pp. 174–181.

[14] J.-F. Chang, S.-C. Chu, J. F. Roddick, and J.-S. Pan, "A parallel particle swarm optimization algorithm with communication strategies," *Journal of Information Science and Engineering*, vol. 21, no. 4, pp. 809–818, 2005.

[15] S.-C. Chang and J.-S. Pan, "Intelligent parallel particle swarm optimization algorithm," in *Parallel Evolutionary Computations*, N. Nedjah, E. Alba, and L. de Macedo Mourelle, Eds. Studies in Computational Intelligence, Springer Berlin, 2006, vol. 22, pp. 159–175.

[16] M. El-Abd and M. S. Kamel, "A hierarchal cooperative particle swarm optimizer," in *Proc. IEEE swarm intelligence symposium*, 2006, pp. 43–47.

[17] G. Yen and M. Daneshyari, "Diversity-based information exchange among multiple swarms in particle swarm optimization," in *Proc. IEEE Congress on Evolutionary Computation*, 2006, pp. 6150–6157.

[18] M. Belal and T. El-Ghazawi, "Parallel models for particle swarm optimizers," *International Journal on Intelligent Cooperative Information Systems*, vol. 4, no. 1, pp. 100–111, 2004.

[19] E. S. Peer, F. van den Bergh, and A. P. Engelbrecht, "Using neighbourhood with guaranteed convergence pso," in *Proc. IEEE Swarm Intelligence Symposium*, 2003, pp. 235–242.

[20] B. Niu, Y. Zhu, and X. He, "Multi-population cooperative particle swarm optimization," in *Proc. European Conference on Artificial Life*, M. C. et al., Ed., 2005, pp. 874–883.

[21] J. J. Liang and P. N. Suganthan, "Dynamic multi-swarm particle swarm optimizer," in *Proc. IEEE Swarm Intelligence Symposium*, 2005, pp. 124–129.

[22] ——, "Dynamic multi-swarm particle swarm optimizer with local search," in *Proc. IEEE Congress on Evolutionary Computation*, 2005, pp. 522–528.

[23] R. Brits, A. P. Engelbrecht, and F. van den Bergh, "A niching particle swarm optimizer," in *Proc. 4th Asia-Pacific Conference on Simulated Evolution and Learning*, 2002.

[24] F. van den Bergh, "An analysis of particle swarm optimizer," Ph.D. dissertation, Department of Computer Science, Univeristy of Pretoria, South Africa, 2002.

[25] X. Li, "Adaptevily choosing neighbourhood bests using species in a particle swarm optimizer for multimodal function optimization," in *Proc. The Genetic And Evolutionary Computation Conference*, 2004, pp. 105–116.

[26] S. Bird and X. Li, "Enhancing the robustness of a speciation-based pso," in *Proc. IEEE Congress on Evolutionary Computation*, 2006, pp. 3185–3192.

[27] J.-H. Seo, C.-H. Im, C.-G. Heo, J.-K. Kim, H.-K. Jung, and C.-G. Lee, "Multimodal function optimization based on particle swarm optimization," *IEEE Transactions on Magnetics*, vol. 42, no. 4, pp. 1095–1098, 2006.

[28] T. Blackwell and J. Branke, "Multi-swarm optimization in dynamic environments," in *Applications in Evolutionary Computing*, G. R. Raidl, Ed. LNCS, Springer-Verlag, 2004, pp. 19–26.

[29] S. Jansen and M. Middendorf, "A hierachical particle swarm optimizer for dynamic optimization problems," in *Proc. Application of Evolutionary Computing*, vol. 3005, 2004, pp. 513–524.

[30] ——, "A hierachical particle swarm optimizer," in *Proc. Application of Evolutionary Computing*, vol. 3005, 2003, pp. 770–776.

[31] ——, "A hierachical particle swarm optimizer for noisy and dynamic environments," *Genetic Programming and Evolvable Machines*, vol. 7, no. 4, pp. 329–354, 2006.

[32] D. Parrott and X. Li, "A particle swarm model for tracking dynamic peaks in a dynamic environment using speciation," in *Proc. IEEE Congress on Evolutionary Computation*, vol. 3, 2004, pp. 98–103.

[33] ——, "Locating and tracking multiple dynamic optima by a particle swarm model using speciation," *IEEE Transactions on Evolutionary Computation*, vol. 10, no. 4, pp. 440–458, 2006.

[34] T. Blackwell and J. Branke, "Multiswarms, exclusion, and anti-convergence in dynamic environments," *IEEE Transactions on Evolutionary Computation*, vol. 10, no. 4, pp. 459–472, 2006.

[35] K. E. Parsopoulos, D. K. Tasoulis, and M. N. Vrahatis, "Multiobjective optimization using parallel vector evaluated particle swarm optimization," in *Proc. International Conference on Artificial Intelligence and Applications*, 2004, pp. 823–828.

[36] K. E. Parsopoulos and M. N. Vrahatis, "Recent approaches to global optimization problems through particle swarm optimization," *Journal of Natural Computing*, vol. 1, pp. 235–306, 2002.

[37] C. Chow and H. Tsui, "Autonomous agent response learning by a multi-species particles swarm optimization," in *Proc. Congress on Evolutionary Computation*, 2004, pp. 778–785.

[38] G. Toscano and A. C. C. Coello, "Using clustering techniques to improve the performance of a multi-objective particle swarm optimizer," in *Proc. Genetic and Evolutionary Computation Conference*, 2004, pp. 225–237.

[39] S. Mostaghim, J. Branke, and H. Schmeck, "Multi-objective particle swarm optimization on computer grids," in *Proc. Genetic and Evolutionary Computation Conference*, 2007, pp. 869–875.

[40] Y. Shi and R. A. Krohling, "Co-evolutionary particle swarm optimization to solve min-max problems," in *Proc. Congress on Evolutionary Computation*, vol. 2, 2002, pp. 1682–1687.

[41] R. A. Krohling, F. Hoffmann, and L. S. Coello, "Co-evolutionary particle swarm optimization to solve min-max problems using gaussian distribution," in *Proc. Congress on Evolutionary Computation*, vol. 1, 2004, pp. 959–964.

[42] F. van den Bergh and A. P. Engelbrecht, "Training product unit neural networks using cooperative particle swarm optimisers," in *Proc. IEEE International Joint Conference on Neural Networks*, vol. 1, 2001, pp. 126–131.

[43] T. Blackwell, "Swarm music: Improvised music with multi-swarms," in *Proc. Symposium on Artificial Intelligence and Creativity in Arts and Science*, 2003, pp. 41–49.

[44] A. M. Abdelbar, S. Ragab, and S. Mitri, "Co-evolutionary particle swarm optimization applied to the 7x7 seega game," in *Proc. IEEE International Joint Conference on Neural Networks*, vol. 1, 2004, pp. 243–248.

[45] A. Asmara, R. A. Krohling, and F. Hoffmann, "Parameter tuning of a computed-torque controller for a 5 degree of freedom robot arm using co-evolutionary particle swarm optimization," in *Proc. IEEE Swarm Inteligence Symposium*, 2005, pp. 162–168.

[46] J. G. Vlachogiannis and K. Y. Lee, "Determining generator contributions to transmission system using parallel vector evaluated particle swarm optimization," *IEEE Transactions on Evolutionary Computation*, vol. 20, no. 4, pp. 1765–1774, 2005.

[47] M. El-Abd and M. S. Kamel, "A taxonomy of cooperative search algorithms," in *Proc. 2nd International Workshop on Hybrid Metaheuristics, LNCS*, vol. 3636, 2005, pp. 32–41.

[48] M. Toulouse, T. G. Craninc, and M. Grendeau, "Communication issues in desiginig cooperative multi-thread parallel searches," in *Meta-Heuristics: Theory and Applications*, 1996, pp. 501–522.

[49] M. Lovbjerg, T. K. Rasmussen, and T. Krink, "Hybrid particle swarm optimiser with breeding and subpopulations," in *Proc. Genetic and Evolutionary Computation Conference*, vol. 1, 2001, pp. 469–476.

[50] C. Blum and A. Roli, "Metaheuristics in combinatorial optimization: Overview and conceptual comparison," *ACM Computing Surveys*, vol. 35, no. 3, pp. 268–308, 2003.

[51] M. El-Abd and M. S. Kamel, "Multiple cooperating swarms for non-linear function optimization," in *Proc. 2nd Workshop on Swarm Intelligence and Patterns*, 2005, pp. 999–1008.

[52] CEC05 benchmark functions. [Online]. Available: http://staffx.webstore.ntu.edu.sg/MySite/Public.aspx?accountname=epnsugan

[53] R. C. Eberhart, P. Simpson, and R. Dobbins, *Computational Intelligence*. PC Tools: Academic, 1996, ch. 6, pp. 212–226.

[54] J. Kennedy and R. Mendes, "Population structure and particle swarm performance," in *Proc. of IEEE Congress on Evolutionary Computation*, vol. 2, 2002, pp. 1671–1676.

[55] P. Larranaga and J. A. Lozano, *Estimation of Distribution Algorithms. A New Tool for Evolutionary Computation*. Kluwer, 2002.

[56] S. Rudolph and M. Koppen, "Stochastic hill climbing with learning by vectors of normal distributions," in *First on-line Workshop on Soft Computing (WSC1)*, 1996, pp. 60–70.

[57] I. Servet, L. Trave-Massuyes, and D. Stern, "Telephone network traffic overloading diagnosis and evolutionary computation technique," in *Artificial Evolution. Springer-Verlag, LNCS 1363*, 1997, pp. 137–144.

[58] M. Sebag and A. Ducoulombier, "Extending population-based incremental learning to continuous search spaces," in *Proc. of Parallel Problem Solving from Nature*, 1999, pp. 418–427.

[59] M. Gallagher, M. Frean, and T. Downs, "Real-valued evolutionary optimization using a flexible probability density estimator," in *Proc. of Genetic and Evolutionary Computation Conference*, vol. 1, 1999, pp. 840–846.

[60] M. Iqbal and M. A. M. de Oca, "An estimation of distribution particle swarm optimization algorithm," in *Proc. of the Fifth International Workshop on Ant Colony Optimization and Swarm Intelligence*, M. Dorigo, L. M. Gambardella, M. Birattari, A. Martinoli, R. Poli, and T. Sttzle, Eds., 2006, pp. 72–83.

[61] K. Socha and M. Dorigo, "Ant colony optimization for continuous domains," Universitie Libre de Bruxelles, Tech. Rep. TR/IRIDIA/2005-037, 2005.

[62] Y. Zhou and J. Jin, "Eda-pso - a new hybrid intelligent optimization algorithm," in *Proc. of the Michigan University Graduate Student Symposium*, 2006.

[63] M. Pelikan, D. E. Goldberg, and F. Lobo, "A survey of optimization by building and using probabilistic models," *Computational Optimization and Applications*, vol. 21, no. 1, pp. 5–20, 2002.

[64] L. delaOssa, J. Gamez, and J. Puerta, "Initial approaches to the application of island-based parallel edas in continuous domains," *Journal of Parallel and Distributed Computing*, vol. 66, no. 8, pp. 991–1001, 2006.

[65] D. Bratton and J. Kennedy, "Defining a standard for particle swarm optimization," in *Proc. IEEE Swarm Intelligence Symposium*, 2007, pp. 120–127.

[66] T. Hiroyaso, M. Miki, M. Sano, H. Shimosaka, S. Tsutsui, and J. Dongarra, "Distributed probabilistic model-building genetic algorithm," in *Proc. of Genetic and Evolutionary Computation Conference*, 2003, pp. 1015–1028.

[67] C. W. Ahn, D. E. Goldberg, and R. S. Ramakrishna, "Multiple-deme parallel estimation of distribution algorithms: Basic framework and applications," in *Proc. of International Con-*

ference on Parallel Processing and Applied Mathematics, Lecture Notes in Computer Science, vol. 3019, Springer-Verlag, 2004, pp. 544–551.

[68] L. delaOssa, J. Gamez, and J. Puerta, "Migration of probability models instead of individuals: An alternative when applying the island models to edas," in *Proc. of Parallel Problem Solving from Nature*, 2004, pp. 242–252.

[69] H. Mhlenbein, "The equation for response to selection and its use for prediction," *IEEE Transactions on Evolutionary Computation*, vol. 5, no. 3, pp. 303–346, 1998.

[70] J. Madera, E. Alba, and A. Ochoa, "A parallel island model for estimation of distribution algorithms," in *Towards a New Evolutionary Comoputation, Advances in the Estimation of Distribution Algorithms, Studies in Fuzziness and Soft Computing*, J. A. Lozano, P. Larraaga, I. Inza, and E. Bengoetxea, Eds. Springer-Verlag, 2006, vol. 192, pp. 159–186.

[71] J. Schwarz, J. Jaros, and J. Ocenasek, "Migration of probabilistic models for island-based bivariate eda algorithm," in *Genetic and Evolutionary Computational Conference*, vol. 1, 2007, pp. 631–631.

[72] J. Jaros and J. Schwarz, "Parallel bmda with probability model migration," in *Proc. IEEE Congress on Evolutionary Computation*, 2007, pp. 1059–1066.

[73] M. Clerc, *Particle swarm optimization.* London: ISTE.M, 2006.

[74] A parameter free pso, tirbes-d. [Online]. Available: http://clerc.maurice.free.fr/pso/Tribes/TRIBES-D.zip

[75] V. Betz, J. Rose, and A. Marquardt, *Architecture and CAD for Deep-Submicron FPGAs.* Norwell, MA: Kluwer Academic Publishers, 1999.

[76] C. Mulpuri and S. Hauck, "Runtime and Quality Tradeoffs in FPGA Placement and Routing," in *Proc. ACM Intl. Symp. on FPGAs*, 2001, pp. 29–36.

[77] A. Salman, A. Imtiaz, and S. Al-Madani, "Discrete Particle Swarm Optimization for Heterogeneous Task Assignment Problem," in *World Multiconference on Systemics, Cybernetics and Informatics (SCI 2001)*, 2001, pp. 83–91.

[78] E. Ozcan and C. K. Mohan, "Particle swarm optimization: Surfing the waves," in *Proc. IEEE Congress on Evolutionary Computation*, vol. 3, 1999, pp. 1939–1944.

[79] M. Clerc, "Discrete Particle Swarm Optimization," in *New Optimization Techniques in Engineering*. Springer-Verlag, 2004.

[80] K.-P. Wang, L. Huang, C.-G. Zhou, and W. Pang, "Particle Swarm Optimization for Traveling Salesman Problem," in *Proc. IEEE International Conference on Systems, Man, and Cybernetics*, vol. 3, 2003, pp. 1583–1585.

[81] X. H. Zhi, X. L. Xing, Q. X. Wang, L. H. Zhang, X. W. Yang, Z. C. G., and Y. C. Laing, "A Disrete PSO Method for Generalized TSP Problem," in *Proc. IEEE International Conference on Systems, Man, and Cybernetics*, vol. 4, 2004, pp. 2378–2383.

[82] Y. Xue, Q. Yang, and J. Feng, "Improved Particle Swarm Optimization Algorithm for Optimum Steelmaking Charge Plan Based on the Pseudo TSP Solution," in *Proc. IEEE International Conference on Systems, Man, and Cybernetics*, vol. 9, 2005, pp. 5452–5457.

[83] C. Wang, J. Zhang, J. Yang, C. Hu, and J. Liu, "A Modified Particle Swarm Optimization Algorithm and its Applications for Solving Travelling Salesman Problem," in *Proc. IEEE International Conference on Neural Networks and Brain*, vol. 2, 2005, pp. 689–694.

[84] T. Liu and J. Wang, "A Discrete Particle Swarm Optimizer for Graphic Presentation of GMDH Network," in *Proc. IEEE International Conference on Systems, Man, and Cybernetics*, vol. 3, 2005, pp. 2329–2333.

[85] X. Zeng, Y. Zhu, L. Nan, K. Hu, B. Niu, and X. He, "Solving Weapon-Target Assignment Problem Using Discrete Particle Swarm Optimization," in *Proc. IEEE World Congress on Intelligent Control and Automation*, vol. 1, 2006, pp. 3562–3565.

[86] Z. Lian, X. Gu, and B. Jiao, "A Dual Similar Particle Swarm Optimization Algorithm for Job-Shop Scheduling With Penalty," in *Proc. IEEE World Congress on Intelligent Control and Automation*, vol. 2, 2006, pp. 7312–7316.

[87] Q. Pan, F. Tasgetiren, and Y. Liang, "A Discrete Particle Swarm Optimization Algorithm for Single Machine Total Earliness and Tardiness Problem With a Common Due Date," in *Proc. IEEE Congress on Evolutionary Computation*, 2006, pp. 3281–3288.

[88] V. G. Gudise and G. K. Venayagamoorthy, "Swarm Intelligence for Digital Circuits Implementation on Field Programmable Gate Arrays Platforms," in *NASA/DoD Conference on Evolvable Hardware*, 2004, pp. 83–91.

[89] M. Neethling and A. Engelbrecht, "Determining RNA Secondary Structure using Set-based Particle Swarm Optimization," in *Proc. IEEE Congress on Evolutionary Computation*, 2006, pp. 1670–1677.

Publications from this thesis

Accepted and Published Journals

- Mohammed El-Abd and Mohamed S. Kamel. "A Taxonomy for Cooperative Particle Swarm Optimizers". The International Journal on Computational Intelligence Research (IJCIR), *special issue on PSO*, vol. 4, issue 2, 2008.

Submitted Journals

- Mohammed El-Abd and Mohamed S. Kamel. "A Heterogeneous Cooperative Particle Swarm Optimizer with Migrated Probability Models". Submitted to Swarm Intelligence, 2008.
- Mohammed El-Abd and Mohamed S. Kamel. "An Empirical Study on Cooperative Particle Swarm Optimizers". Submitted to Swarm Intelligence, *special issue on PSO*, 2008.
- Mohammed El-Abd, Hassan Hassan, Mohab Anis, Mohamed S. Kamel and Mohamed El-Masry. "Discrete Cooperative Particle Swarm Optimization for FPGA Placement". Submitted to the Journal of Applied Soft Computing, Elsevier, 2007.

Book Chapters

- Mohammed El-Abd and Mohamed S. Kamel. "Cooperative Particle Swarms Optimizers: A Powerful and Promising Approach". In Springer-Verlag Studies in Computational Intelligence (SCI) Series, vol 34, Abraham, Ajith; Grosan, Crina; Ramos, Vitorino (Eds.), Stigmergic Optimization, pp. 239-260, 2006.

Conference Papers

- Mohammed El-Abd and Mohamed S. Kamel. "A Particle Swarm Optimizer with Varying Bounds". Proceedings of the IEEE Congress on Evolutionary Computation CEC, pp. 4757-4761, Singapore, September 2007.

- Mohammed El-Abd and Mohamed S. Kamel. "On The Convergence of Information Exchange Methods in Multiple Cooperating Swarms". Proceedings of the IEEE Congress on Evolutionary Computation CEC, pp. 3797-3801, Vancouver, July 2006.

- Mohammed El-Abd and Mohamed S. Kamel. "A Hierarchal Cooperative Particle Swarm Optimizer". Proceedings of the IEEE Swarm Intelligence Symposium SIS06, pp. 43-47, Indianapolis, May 2006.

- Mohammed El-Abd and Mohamed S. Kamel. "A Taxonomy of Cooperative Search Algorithms". Proceedings of the 2nd International Workshop on Hybrid Meta-Heuristics, LNCS 3636, pp. 32-41, Barcelona, Spain, August 2005.

- Mohammed El-Abd and Mohamed S. Kamel. "Information Exchange in Multiple Cooperating Swarms". Proceedings of the IEEE Swarm Intelligence Symposium SIS05, pp. 138-142, Pasadena, CA, June 2005.

- Mohammed El-Abd and Mohamed S. Kamel. "Factors Governing the Behavior of Multiple Cooperating Swarms". Proceedings of the Genetic and Evolutionary Computation Conference GECCO05, vol. 1, pp. 269-270, Washington DC, June 2005.

- Mohammed El-Abd and Mohamed S. Kamel. "Multiple Cooperating Swarms for Non-Linear Function Optimization". Proceedings of the IEEE 4th International Workshop on Soft Computing as Transdisciplinary Science and Technology WSTST 05, 2nd workshop on Swarm Intelligence and Patterns SIP05, pp. 999-1008, Japan, May 2005.