CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. (2012)
Published online in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/cpe.2910

SPECIAL ISSUE PAPER

Evaluation of asynchronous multi-swarm particle optimization on several topologies

Arion de Campos Jr., Aurora T. R. Pozo and Elias P. Duarte Jr.*,†

Department of Informatics, Federal University of Parana, Curitiba, PR, Brazil

SUMMARY

Particle swarm optimization is a population-based stochastic optimization technique that is easy to implement and has been successfully applied in many areas. However, its performance often deteriorates as the dimensionality of the problem increases. Recently, parallel strategies based on multiple swarms (multi-swarm) have been investigated as an alternative to overcome this problem. In this paper, we evaluate the impact of the topology on multi-swarm systems, considering that swarms are independent, and interact by means of particle migration. We focus on asynchronous communication, that is, only when an improvement occurs on the best particle that the solution migrates among swarms. The goal is to check how different communication strategies affect the parallel execution of the optimization tasks. Several different topologies and communication strategies have been evaluated, including broadcast and gossip on fully connected networks, unidirectional and bidirectional rings, hypercubes, and a dynamic topology. Extensive experimental results were obtained and are reported using several traditional benchmark functions. We evaluated the impact of the topologies in terms of the number of iterations and the communication overhead. With the results, a ranking of the different topologies is presented. The impact of the number of swarms on the optimization process is also evaluated. Copyright © 2012 John Wiley & Sons, Ltd.

Received 15 January 2012; Revised 21 June 2012; Accepted 6 July 2012

KEY WORDS: evolutionary computation; distributed particle swarm optimization; multi-swarm

1. INTRODUCTION

Since Particle Swarm Optimization (PSO) was first introduced [1], many applications have been developed using this stochastic metaheuristic optimization technique [2–4]. PSO is based on the simulation of social behavior observed in animals and insects, and it is considered to be a simple and efficient technique [5]. In comparison with Evolutionary Algorithms (EAs), PSO is an algorithm that quickly finds good results; however, its performance degrades as the problem size increases [6]. The computational cost increases for solving problems with high dimensionality.

The high computational cost for solving complex optimization problems has motivated the development of parallel strategies to speed up the solutions. Several factors have contributed to reinforce the relevance of parallel optimization algorithms, including evidences of higher efficiency and larger diversity maintenance [7]. In current systems, there is an increasing availability of processors and memory; several technologies have made it cheap and feasible to deploy parallel solutions on a variety of platforms.

*Correspondence to: Elias P. Duarte Jr., Department of Informatics, Federal University of Parana, Rua Cel. Francisco H. dos Santos, 100, Centro Politécnico, Jardim das Américas, P.O.Box: 19081, Curitiba, PR, 81531-980, Brazil.
†E-mail: elias@inf.ufpr.br

Multi-swarm algorithms are parallel and distributed optimization techniques whose individuals are co-evolving swarms, which are independent and interact by means of particle migration policies. El-Abd and Kamel [8] proposed a taxonomy for cooperative PSO algorithms. The authors bring together a comprehensive survey of several cooperative PSO models, identifying the basic design decisions that differentiate these models. A classification scheme is introduced to the different cooperative strategies based on the approach used by the multiple swarms to share information. Four aspects are considered: *which* information is shared, *when* information is shared, *how* information is shared, and finally *what* is done with the shared information. Observing the several possible combinations, there is a variety of work based on multiple swarms; however, most of them address problems with up to 30 dimensions, the cooperation is synchronous, that is, based on a fixed number of iterations, and the impact of the topology interconnecting the swarms is ignored.

This work evaluates the impact of the topology on multi-swarm optimization. Swarms use asynchronous information sharing. The cooperation consists in running multiple swarms on independent processors, which are interconnected and communicate with each other using different strategies. We evaluate how efficiently these different strategies allow asynchronous multi-swarm optimization solutions to explore the search space and obtain better solutions faster. The following topologies/communication strategies were adopted for the swarms to share the social component: broadcast, unidirectional ring, bidirectional ring, gossip to one-neighbor, gossip to $\log(N)$-neighbors, and the hypercube. We also employ a dynamic topology to interconnect the swarms, which is based on the Frankenstein algorithm [9] and starts as a fully connected broadcast network to a bidirectional ring as the optimization process evolves. To the best of our knowledge, this topology has never been used before to run multi-swarm optimization. The rationality behind this dynamic topology is that, during the first iterations, the swarms are given the opportunity to find and exchange good quality solutions among themselves, and later, the connectivity is decreased to enhance the independent exploration of the search space by each individual swarm.

Extensive experimental results were obtained and are reported using several traditional benchmark functions. Problems with up to 100 dimensions were first solved using centralized optimization to obtain reference results. The functions are then solved with multiple swarms connected by different topologies. The results are statistically analyzed. For each topology, the convergence property is maintained at least for one instance. We also evaluated the number of messages employed by each different alternative. On the basis of the results, we rank the different topologies. The impact of the number of swarms on the optimization process is also evaluated.

The remainder of this paper is organized as follows. Section 2 presents an overview of PSO and points to related work in Multi-Swarm PSO (MSPSO). Section 3 describes our proposed strategy for the execution of multi-swarms on several different topologies using asynchronous communication. Section 4 presents our empirical evaluation, with a detailed description of the simulation environment, the benchmark functions, parameter settings and results. Finally, Section 5 concludes the paper and presents future research directions.

2. PARTICLE SWARM AND MULTI-SWARM OPTIMIZATION

Particle swarm optimization is a population-based stochastic optimization technique developed by Eberhart and Kennedy in 1995 [1], inspired by the social behavior of groups of animals, like flocks of birds and schools of fish. PSO shares many similarities with evolutionary computation techniques such as Genetic Algorithms (GA). The system is initialized with a population of random solutions and searches for optima by updating generations. However, unlike GA, PSO has no evolution operators such as crossover and mutation. In the algorithm, each individual is called a particle and behaves like a bird (or fish) in the flock (school) searching for food or fleeing from a predator. Each particle has a position and a velocity used to explore the search space of the problem. A position is linked to the solution of the problem and represents a potential solution. To achieve its goal, a particle learns from its own experiences (information or individual component) and also learns from the group (information or social component). The best solution found by the particle itself is called *pbest* (personal best) and the best solution found by the group is called *gbest* (global value).

DOI: 10.1002/cpe

```
1: for each particle i do
2:   initialize x_i, v_i;
3: end;
4: while the maximum number of iterations or the minimum error is not attained do
5:   for each particle i do
6:     compute current fitness value f(x_i);
7:     if f(x_i) is better than f(pbest_i) then
8:       pbest_i ← x_i;
9:     end;
10:  end;
11:  gbest_i ← bestOf(pbest_i), i=1 to N;
12:  for each particle i do
13:    compute the velocity v_i according to equation (1);
14:    update position x_i according to equation (2);
15:  end;
16: end.
```

Figure 1. The classical particle swarm optimization algorithm.

The pseudo-code of the classical PSO algorithm is shown in Figure 1. In the Figure, the ith solution (particle) is represented by x_i and its velocity is represented by v_i. Initially the algorithm sets random values for each particle and its velocity, within the limits defined for each function. Next, the algorithm runs a loop that computes the fitness for each particle. The value of the best solution is stored in local variable *pbest* (in the code, $pbest_i$). The best global fitness is then computed by checking all *pbest* values and is stored in variable *gbest* (global best). After finding *pbest* and *gbest*, each particle updates its velocity and position according to Equations (1) and (2) as follows.

$$v_i = \varpi v_i + c_1 \mathrm{rand}()(pbest_i - x_i) + c_2 \mathrm{rand}()(gbest_i - x_i) \tag{1}$$

$$x_i = x_i + v_i \tag{2}$$

In these equations, c_1 and c_2 are constants that define the learning factor and rand() is a random number between 0 and 1. v_i is the particle velocity; $pbest_i$, the best value found by particle i itself; x_i corresponds to the position of the particle being manipulated and *gbest* is the best solution found by the group. w denotes the weight of inertia (tendency to maintain the same particle velocity) in the updated velocity. To prevent an excessive speed, the velocity is limited to a maximum (which is function-dependent, and is called $vmax$). If the speed exceeds this value, it is truncated back to $vmax$. At the end of a number of iterations or some other stop criterion, the best solution is presented as a result.

A swarm consists of a set of particles moving in an n-dimensional search space. As noted by Equation (1), its velocity is conditioned by the best solution found by the particle and the best position achieved by its neighbors. The definition of the best solution in a neighborhood depends on the neighborhood topology implemented. Each topology determines how the particles exchange the social component. Kennedy [10] considers four topologies: ring, star, complete graph and arbitrary.

Several other topologies exist and have been investigated, such as the tree network topology introduced by Janson and Middendorf [11]. In this topology, all particles are arranged in a tree. A particle is influenced by its own best position so far (*pbest*) and by the best position of the particle that is its parent in the tree.

In [9], de Oca *et al*. present a dynamic topology, called Frankenstein. This topology starts as fully connected and, as the optimization process evolves, the connectivity decreases until it becomes a ring. The authors argue that using a highly connected topology during the first iterations gives to the algorithm the opportunity to find good quality solutions early in a run. As the topology connectivity decreases, the risk to get trapped in a local optimum is reduced, hence, the exploration is enhanced. Assuming that the swarm is composed of n particles, the topology is scheduled to be modified at each k iterations (with $k \geqslant n$). At every $k/(n-3)$ iterations m edges are removed; these edges follow arithmetic regression pattern of the form $n-2, n-3, \ldots, 2$. m nodes are then removed one edge per node. The edge to be removed is chosen randomly from the edges that do not belong to the exterior ring, which is predefined.

DOI: 10.1002/cpe

Parallel optimization with multiple swarms

Recent advances in computer and network technologies have led to the development of a variety of parallel optimization algorithms [7]. Population-based Algorithms are specially suited to work in parallel, and several parallel EAs have been proposed [12].

Parallel algorithms are usually classified as either coarse or fine grained. This classification relies on the computation/communication ratio. If the ratio is high (roughly speaking: the algorithm presents more computation than communication), we say the algorithm is coarse-grained. Otherwise, if the ratio is low (more communication than computation), then we say it is a fine-grained parallel EA. Coarse-grained EAs are also known as *distributed EAs* or *island EAs*, and an example of fine-grained EAs are the cellular GAs [13].

Multi-swarm optimization consists of the division of the population of particles into subpopulations collectively called multi-swarms, and represent a model of cooperative optimization. The same classification used for EAs can be used for multi-swarm algorithms. The first work that proposed the use of multiple swarms in a cooperative model of optimization based on PSO was proposed by Van der Bergh and Engelbrecht [14]. In that work, two algorithms were proposed: Cooperative Particle Swarm Optimizer (CPSO-Sk and CPSO-Hk). The first algorithm splits the solution vector into k swarms (smaller vectors), and each swarm is optimized independently. At each iteration, the solution vector is rebuilt taking into consideration the best particle of each swarm. To update each particle, the best particle of each swarm by itself and the best particles of all swarms are considered. The second algorithm is a hybrid implementation that combines the standard PSO algorithm with the CPSO-Sk algorithm. The purpose is to combine the best features of both algorithms in one solution. The idea is to create a cycle where some particles of an algorithm are replaced by the best particles of the other algorithm. As a drawback, both CPSO algorithms are dependent on the type of function being optimized, as noted by Li and Yao [15, 16]. Because swarms are strongly correlated, as each swarm contains a part of the solution vector, functions that cannot be separated result in degraded performance for both algorithms. Another issue to note is the computational cost, as both local and global best particles are constantly evaluated; the cost is substantially high.

Another work involving multi-swarms was presented by Liang and Suganthan [17] in which the authors present the Dynamic Multi-Swarm-PSO algorithm. Dynamic Multi-Swarm-PSO is based on classical PSO, which is run locally at each system node, which also employs its neighborhood in the search for the solution. The topology that connects swarms, called 'neighborhood', presents the following two important features: (i) it is based on small- sized swarms; and (ii) swarms are randomly regrouped periodically. After a predefined number of iterations, the population is regrouped randomly, and the search is restarted using a new configuration of small swarms. The authors report that a small number of particles per swarm can achieve better results for simple problems, whereas for complex problems, employing a small neighborhood of larger swarms yields better results. In this configuration, the convergence speed decreases and the diversity increases. Evaluation results were reported for benchmark functions, and the authors concluded that the best results are obtained in a configuration with 30 particles, three particles per swarm, which are randomly regrouped every five iterations. Results are given for problems with up to 10 dimensions.

An approach that takes advantage of some dynamic aspects of certain optimization problems is proposed by Geng and Zhu [3], which divides the population into three swarms of particles. The first swarm runs the standard PSO, the second swarm runs a variation of the original PSO in which it is possible that a particle deviates from the local optimum, thus not necessarily going to the best position. In the third swarm, each particle 'flies' freely, without following the best particle. The best particle is evaluated at each iteration. A fully connected topology is used and alternative topologies are not considered. In [2], the Cooperative Co-evolutionary PSO algorithm, it is proposed that it incorporates a mutation operator and exposes an alternative method for the selection of particles of each swarm that will be the next leader (*gbest*). This method is based on a random choice, in which there is a probability of an individual with poor fitness to be chosen. This assumption is intended to increase the chance of escaping from local optima. The algorithm is evaluated for only two benchmark functions.

DOI: 10.1002/cpe

Another work addressing dynamic aspects of optimization problems is presented by Xiangwei and Hong [4]. The authors note that PSO does not perform well for functions where the global optimum is modified over time. This limitation depends on the behavior of the function to be optimized. To avoid this limitation, the authors propose Different Topology Multi-Swarm PSO. Two swarms are created—G-Swarm and L-Swarm—of which the first is fully connected, and the second consists of isolated particles. The G-swarm's best particle (*gbest*) influences the entire swarm, whereas in the L-swarm, each particle is influenced only by its best position (*pbest*) found so far. After a predefined period, the best particles are exchanged between the two swarms. A mutation operator is applied to the particles to avoid stagnation. The aim is to combine the fast convergence of the G-swarm with the preservation of diversity of L-swarm for solving dynamic problems. The authors evaluate the algorithm with several benchmark functions and report good results. The work only addresses problems with 10 dimensions and does not evaluate the performance of the algorithm when the particles are exchanged at predefined periodic time intervals.

Vanneschi *et al.* [18] introduce four algorithms for solving optimization problems, each algorithm with 10 swarms and 10 particles per swarm. One is a hybrid algorithm, combining PSO and GA, that periodically applies a set of mutation and crossover operators. The second algorithm is a variation of the first one, and includes a repulsive component, in which particles are attracted by the best positions of their own swarms and move away from the best positions of other particles. A third algorithm, more closely related to the contribution of the current work, called MSPSO uses a set of swarms, each of which solves problems independently. This set is connected using a ring topology and at every 10 iterations, an interaction occurs between two swarms: a swarm sends 20% of its best particles to the neighbor swarm. The swarm that receives the particles replaces 20% of its worst particles with the incoming particles. A fourth algorithm presented by the authors is a variation of the previous algorithm, which includes a repulsive component. The algorithms are evaluated using different benchmark functions, ranging from simple functions to those considered more complex. Results presented are good for some functions and not so good for others. The authors propose new benchmark functions with intermediate complexity. It should be noted that the work only deals with problems with 20 dimensions, a single topology (in this case, the ring topology) is adopted, and there is no evaluation of the impact of the variation of the time interval used to exchange particles on the performance of the algorithms.

El-Abd and Kamel [8] proposed a taxonomy to identify and classify the optimization strategies on the basis of CPSO. According to the authors, to create a cooperative solution on the basis of PSO, various aspects have to be considered to obtain a better performance. These aspects were specified in the form of the following four questions:

(1) *Which information to exchange?* In multi-swarm optimization, the most common information exchanged is the *gbest*, or information about the best particles and the leaders. However, the *gbest* of each swarm can represent different information. For example, the CPSO [14] relies on splitting the space (solution vector) into sub-spaces (smaller vectors) where each subspace is optimized using a different swarm. The overall solution vector is built using the solutions found by the best particle of each swarm. This approach was originally introduced using GA [19]. On the other hand, the "Island model" is a group of parallel optimization processes, which occasionally send individuals to one another to help spread newly discovered fit areas of the space. All *gbest* represent the same kind of information. This approach has been widely applied to parallel GA [12]. However, in GA usually the number of individuals that migrates from one island to another is greater than one. Figure 2 depicts these models.
(2) *When to exchange the information?* Basically, there are two communication strategies: synchronous and asynchronous. The synchronous strategy dictates that information is exchanged periodically at iterations scheduled at fixed time intervals. Asynchronous communication is triggered by the occurrence of an event, such as the stagnation of results or after a specified number of iterations.

DOI: 10.1002/cpe

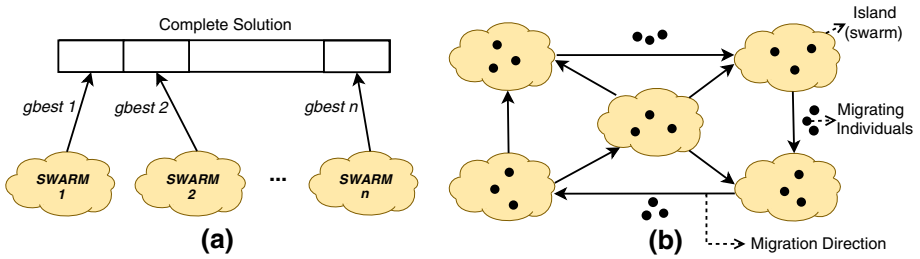

Figure 2. Two approaches for sharing information: (a) Co-evolutionary Model; and (b) Island Model.

(3) *How to share the information?* Communication occurs both between neighbors within a swarm and between particles of different swarms;
(4) *What to do with the exchanged information?* Several actions are possible, such as to upgrade speeds, replace particles or reset swarms.

Despite the intensive research activity and the large number of strategies that have been proposed for multi-swarm optimization, there are several open questions. For instance, to the best of our knowledge, this work is the first to use complex functions to evaluate several different topologies for solving multi-swarm optimization based on asynchronous communication, as described in the next section.

3. ASYNCHRONOUS MULTI-SWARM PARTICLE OPTIMIZATION

In this section, we describe our strategy for executing multiple swarms on several different topologies using asynchronous communication. Particles are organized in swarms (subpopulations) each of which have a fully connected topology. The purpose of adopting this strategy is to allow each population to independently perform the optimization process in an attempt to speed up this process, and also to prevent the convergence to suboptimal solutions [20]. By adopting multiple processors to run the swarms independently, the optimization process can be executed in parallel, which may represent the only feasible strategy for optimizing functions with high dimensionality.

Using the taxonomy introduced in [8], our system model can be defined as follows:

(1) *The information exchanged is the best particle of each swarm.*
(2) *The communication strategy is asynchronous.* The best particle of each swarm is sent only when there is an improvement in the *gbest*;
(3) *The information is shared on different topologies and communication strategies, including:* broadcast, unidirectional ring, bidirectional ring, gossip to one-neighbor, gossip to $\log(N)$-neighbors, the hypercube and the Frankenstein dynamic topology.
(4) *Updating the* gbest*:* The *gbest* received from another swarm is evaluated taking into account the current value kept for *gbest* at the swarm itself. If the received value represents an improvement, then it becomes the current *gbest*.

The particle population is distributed into several swarms. Each swarm executes independently the optimization process. One *gbest* is computed per swarm, and this *gbest* has influence only on the position of the particles of this swarm. This information is then exchanged between the different swarms. For all topologies/communication strategies to share information, the procedure adopted in this paper is as follows:

1. When a swarm improves its *gbest*, one or more neighbors are chosen (depending on the topology employed) as destinations to send the *gbest* and fitness;
2. The destination swarm compares the received information with its own *gbest*/fitness;
3. If the received *gbest* is better, the swarm takes it as the global *gbest* itself, otherwise the information is ignored.

DOI: 10.1002/cpe

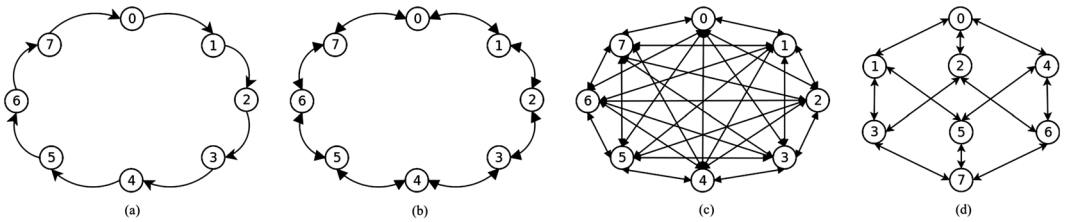

Figure 3. Topologies: (a) unidirectional ring; (b) bidirectional ring; (c) fully connected; and (d) hypercube.

The main goal of this work is to investigate the impact of the topology on multi-swarm performance. The following topologies/communication strategies were adopted for the swarms to share the social component (*gbest*):

1. *Broadcast*: By using broadcast, the social component is sent from a swarm to all other swarms, based on a fully connected topology;
2. *Unidirectional ring*: With the ring topology, the social component is sent from swarm i to its successor in the ring;
3. *Bidirectional ring*: Also with a ring topology, the social component is sent to both the predecessor or successor swarms of i;
4. *Gossip to one-neighbor*: Gossip (also called epidemic) protocols are widely used strategies for information dissemination in computer networks [21]. New information is sent to randomly chosen destinations; the number of such destinations is called the *fanout*. This strategy is based on a fully connected topology, and the fanout is equal to 1;
5. *Gossip to log(N)-neighbors*: Similar to the previous method, on the basis of a fully connected topology, in this case the fanout is $\log(N)$, that is, $\log(N)$ swarms are chosen randomly to receive the social component.
6. *Hypercube*: In the hypercube topology, each swarm is connected to $\log(N)$ other swarms, such that a swarm is adjacent to another if their identifiers differ in only one bit. The social component is sent from a given swarm to its $\log(N)$ neighbor swarms.
7. *Dynamic*: This is the time-varying topology proposed by de Oca *et al.* [9]. Initially, all swarms are fully connected. The connectivity decreases over time and, in the end, each swarm sends its social component to its predecessor and successor neighbors. A swarm sends its social component to all its neighbors.

Figure 3 illustrates four of the topologies: unidirectional ring, bidirectional ring, fully connected and hypercube; vertices represent swarms and edges represent the relationships between the swarms. To the best of our knowledge, the hypercube and the dynamic topologies have never been used before as topologies to connect multi-swarms in PSO.

As Figure 3 depicts, in each topology, eight independent swarms were connected. It is important to note that three different approaches were used to exchange the social component in the fully connected topology: broadcast, gossip to one-neighbor and gossip to $\log(N)$-neighbors.

4. EMPIRICAL EVALUATION

In this section, we describe the experimental results of the performance evaluation of multi-swarm optimization on several topologies.

A set of six benchmark functions were considered in a multidimensional search space. These functions, which are widely used [22], are the following:

(1) F_1: *Shifted Sphere Function*

$$F_1(x) = \sum_{i=1}^{D} z_i^2 + f\_\text{bias}_1, \quad \mathbf{z} = \mathbf{x} - \mathbf{o},\ \mathbf{x} = [x_1, x_2, \ldots, x_D] \tag{3}$$

DOI: 10.1002/cpe

D = 100 dimensions. o = $[o_1, o_2, \ldots, o_D]$: the shifted global optimum.
Properties: unimodal, shifted, separable, scalable.

$$x \in [-100, 100]^D,$$
$$\text{Global optimum: } x^* = o, F_1(x^*) = f\_\text{bias}_1 = -450$$

(2) F_2: *Schwefel's Problem* 2.21

$$F_2(x) = \max_{i=1}\{|\, z_i \,|, 1 \leqslant i \leqslant D\} + f\_\text{bias}_2, \quad \mathbf{z} = \mathbf{x} - \mathbf{o},\ \mathbf{x} = [x_1, x_2, \ldots, x_D] \tag{4}$$

D = 100 dimensions. o = $[o_1, o_2, \ldots, o_D]$: the shifted global optimum.
Properties: unimodal, shifted, non-separable, scalable.

$$x \in [-100, 100]^D,$$
$$\text{Global optimum: } x^* = o, F_2(x^*) = f\_bias_2 = -450$$

(3) F_3: *Shifted Rosenbrocks's Function*

$$F_3(x) = \sum_{i=1}^{D-1} \left(100\left(z_i^2 - z_{i+1}\right)^2 + (z_i - 1)^2\right) + f\_\text{bias}_3, \quad \mathbf{z} = \mathbf{x} - \mathbf{o},\ \mathbf{x} = [x_1, x_2, \ldots, x_D] \tag{5}$$

D = 100 dimensions. o = $[o_1, o_2, \ldots, o_D]$: the shifted global optimum.
Properties: multi-modal, shifted, non-separable, scalable, having a very narrow valley from local optimum to global optimum.

$$x \in [-100, 100]^D,$$
$$\text{Global optimum: } x^* = o, F_3(x^*) = f\_\text{bias}_3 = 390$$

(4) F_4: *Shifted Rastrigin's Function*

$$F_4(x) = \sum_{i=1}^{D} \left(z_i^2 - 10\cos(2\pi z_i) + 10\right) + f\_\text{bias}_4, \quad \mathbf{z} = \mathbf{x} - \mathbf{o},\ x = [x_1, x_2, \ldots, x_D] \tag{6}$$

D = 100 dimensions. o = $[o_1, o_2, \ldots, o_D]$: the shifted global optimum.
Properties: multi-modal, shifted, separable, scalable, local optima's number is huge.

$$x \in [-5, 5]^D,$$
$$\text{Global optimum: } x^* = o, F_4(x^*) = f\_\text{bias}_4 = -330$$

(5) F_5: *Shifted Griewank's Function*

$$F_5(x) = \sum_{i=1}^{D} \frac{z_i^2}{4000} - \prod_{i=1}^{D} \cos\left(\frac{z_i}{\sqrt{i}}\right) + 1 + f\_\text{bias}_5, \quad \mathbf{z} = \mathbf{x} - \mathbf{o},\ \mathbf{x} = [x_1, x_2, \ldots, x_D] \tag{7}$$

D = 100 dimensions. o = $[o_1, o_2, \ldots, o_D]$: the shifted global optimum.
Properties: multi-modal, shifted, non-separable, scalable.

$$x \in [-600, 600]^D,$$
$$\text{Global optimum: } x^* = o, F_5(x^*) = f\_\text{bias}_5 = -180$$

DOI: 10.1002/cpe

(6) F_6: *Shifted Ackley's Function*

$$F_6(x) = -20\exp\left(-0.2\sqrt{\frac{1}{D}\sum_{i=1}^{D} z_i^2}\right) - \exp\left(\frac{1}{D}\sum_{i=1}^{D}\cos(2\pi z_i)\right) + 20 + e + f\_\text{bias}_6,$$
$$\mathbf{z} = \mathbf{x} - \mathbf{o},\ \mathbf{x} = [x_1, x_2, \ldots, x_D] \tag{8}$$

D = 100 dimensions. o $= [o_1, o_2, \ldots, o_D]$: the shifted global optimum.
Properties: multi-modal, shifted, separable, scalable.

$$x \in [-32, 32]^D,$$
$$\text{Global optimum: } x^* = o, F_6(x^*) = f\_bias_6 = -140$$

Each of the functions mentioned were evaluated using seven different topologies/communication strategies: broadcast, dynamic, hypercube, gossip to log(N)-neighbors, bidirectional ring, unidirectional ring, gossip to one-neighbor. A centralized environment running the standard PSO was first executed to obtain a reference (baseline) results. Next, we executed each optimization function using 8, 16 and 32 swarms on the different topologies. We report results on the number of iterations and the communication overhead in terms of the number of messages employed. On the basis of the results obtained, we rank the different topologies using the *Success Performance* evaluation criterium [23]. Finally we also evaluated the impact of the number of swarms on the optimization process.

In the multi-swarm experiments, each swarm starts by locally executing one PSO iteration in order to try to refine the results. If a swarm improves its best result, then a communication event is triggered. According to the topology, the swarm sends its result to other swarms. A swarm that receives the information compares the received results with its own results. If the received result is better, then the received particle becomes the *gbest* of the swarm. Otherwise, the information is ignored. Note that this strategy for sharing the social component among swarms is classified as asynchronous communication. The communication is triggered by an event (in this case, the improvement of the results) rather than by a fixed number of iterations. The simulation completes when the result for each function presents a minimum error or when a maximum number (1,000,000) of iterations is reached. To evaluate the impact of the number of swarms, the number of iterations was fixed at 60,000 as the only stopping criterium.

To obtain experimental results and evaluate multi-swarm optimization on several topologies, we implemented a simulator using SiMulation Programming Language (SMPL) [24]. SMPL is a discrete-event simulation toolkit developed on top the traditional C programming language. The methodology and parameters employed on the simulation are presented next.

Simulation: methodology and parameters

As mentioned earlier, a baseline reference was first obtained by running the classical centralized PSO. To obtain the reference values, each function used 1000 particles, each particle consisting of 100 randomly initialized values in the scope of each function. The values used for the parameters are those traditionally employed [25]. The inertia weight (w) was initialized at 0.9 and was gradually reduced down to 0.4. This was done because by reducing the inertia weight, the result tends to be improved. Constants $c1$ and $c2$ were set to 2, values commonly employed in the literature.

Each particle has its corresponding values within the limits of each dimension. For functions F1, F5 and F6, each run was executed until the best result reached an error of less than 0.001 in relation to the global optimum of each function. For other functions, the stop criterion has an error of less than 0.1.

All functions were executed 30 times, and the best result and the total number of messages transmitted for each run were stored. Note that the lower the number of iterations and messages exchanged, the better the execution is. If, at the end of 1,000,000 iterations the round does not reach the minimum error, we consider that the execution did not converge to the global optimum. This is also recorded.

DOI: 10.1002/cpe

In the distributed simulation, the same parameters were used, with the exception of the number of particles associated with each swarm. A total of 1000 particles were distributed among the swarms. Eight, 16 and 32 swarms were used; the number of particles per swarm was 125 (eight swarms), 62 or 63 (16 swarms) and 31 or 32 (32 swarms). This configuration allowed a fair comparison with the reference baseline results.

When reaching a minimum error is the criterion to stop the execution, it is fairly common to observe executions that do not converge. To evaluate the impact of the number of swarms (henceforth also called 'islands') on the optimization performance, we fixed the number of iterations at 60,000 and varied the number of islands.

The parameters used in the dynamic topology were as follows. The topology starts as a fully connected (complete) graph and after 30,000 iterations it is transformed into a ring topology (with n edges). The total number of edges that have to be eliminated is $n(n3)/2$. Every $\lceil 30000/(\mathrm{n}-3)\rceil$ iterations, m edges are removed, where m follows an arithmetic regression pattern of the form $n-2, n-3, \ldots, 2$. $n=8$, $n=16$ and $n=32$ for 8, 16 and 32 swarms, respectively.

An additional feature incorporated into the distributed simulation was the exchange of the *gbest*. Unlike the reference executions that share memory addresses in a tightly coupled system, an explicit mechanism was needed for the communication among the swarms. According to the topology, each swarm chooses one or more neighbors to send its *gbest* as described in Section 3. The *gbest* exchange is triggered when it improves. This asynchronous communication strategy avoids unnecessary communications in comparison with the synchronous model (exchanges information periodically at every fixed number of iterations).

For each topology, 30 runs were executed. In the next section, results are presented and discussed.

Results

The results shown in Table I correspond to the number of iterations required to reach results with the expected error. The table presents results for all functions and all topologies and communication strategies used. Table II presents the number of messages employed by the multi-swarm strategies. The presented results are the mean and the standard deviation computed on samples obtained from 30 executions for each function. To compare the results, the Friedman test was applied with significance level of 5%. The R toolkit for statistical computing was employed [26]. When the test shows a significant statistical difference, the best averages are highlighted in bold.

Table I. Results for each function for different topologies.

Topology		F1	F2	F3	F4	F5	F6
Reference	Average	**29032.63**	24 did not	2 did not	3 did not	4 did not	**53154.23**
	Std. dev.	**736.30**	converge	converge	converge	converge	**1902.11**
Broadcast	Average	**35243.47**	4 did not	**171623.07**	**110530.30**	**41369.17**	64558.27
	Std. dev.	**913.26**	converge	**14002.91**	**7410.34**	**1187.07**	2199.58
Dynamic	Average	45856.90	**155003.90**	**175149.43**	122454.50	3 did not	73823.73
	Std. dev.	1146.91	**12067.70**	**15072.77**	10316.42	converge	2309.50
Hypercube	Average	41748.60	165302.60	**171973.73**	**115777.20**	5 did not	66448.17
	Std. dev.	1160.68	13754.06	**14531.19**	**8356.63**	converge	1819.39
Gossip	Average	60653.67	164329.17	**169705.43**	**113740.40**	5 did not	65535.33
Log(N)	Std. Dev.	34729.88	10640.63	**17661.80**	**8166.78**	converge	1932.66
Bidirec.	Average	49326.90	**157296.03**	182138.73	3 did not	3 did not	70804.30
Ring	Std. dev.	1015.87	**9972.80**	12178.46	converge	converge	2387.16
Unidirec.	Average	51695.77	**156820.80**	185368.70	4 did not	4 did not	74129.33
Ring	Std. dev.	2270.94	**10579.48**	11114.19	converge	converge	3121.07
Gossip 1	Average	47358.03	**157424.77**	**168320.30**	1 did not	4 did not	69404.40
	Std. Dev.	957.82	**10777.95**	**15718.40**	converge	converge	1551.86

Note: When the test shows a significant statistical difference, the best averages are highlighted in bold.

DOI: 10.1002/cpe

Table II. Communication overhead.

Topology		F1	F2	F3	F4	F5	F6
Broadcast	Average	32596.00	4 did not	1052594.50	2949667.00	**39646.50**	103207.50
	Std. Dev.	1334.85	converge	209003.29	3958764.01	**1613.43**	8002.25
Dynamic	Average	21301.37	194183.13	432354.90	**89871.00**	5 did not	42089.87
	Std. Dev.	596.22	38595.75	97950.53	**14560.72**	converge	3002.75
Hypercube	Average	17106.13	345904.40	613873.33	1273676.13	5 did not	44785.73
	Std. Dev.	555.48	69597.62	141762.07	1677997.76	converge	3251.96
Gossip	Average	15716.67	331350.00	577488.13	1205075.20	5 did not	42109.20
Log(N)	Std. Dev.	439.14	53906.51	156441.61	1588557.85	converge	3120.67
Bidirec.	Average	**8087.27**	**117656.33**	296907.07	3 did not	3 did not	**17559.23**
Ring	Std. Dev.	**256.34**	**18527.19**	59810.58	converge	converge	**1790.63**
Unidirec.	Average	9518.47	**136610.80**	339162.37	4 did not	4 did not	21027.30
Ring	Std. Dev.	280.36	**25730.57**	56223.00	converge	converge	2928.17
Gossip 1	Average	**6965.20**	**114750.97**	**227063.27**	1 did not	4 did not	**15582.77**
	Std. Dev.	**267.03**	**21286.00**	**62592.10**	converge	converge	**1159.16**

Note: When the test shows a significant statistical difference, the best averages are highlighted in bold.

For the Sphere function (F1), the standard PSO provides the best results; however, there is no statistical difference with the broadcast strategy. For Ackey's function (F6), a similar behavior can be observed, explained by the fact that both functions have their results strongly influenced by the position of the best particle. For all other functions, the performance of standard PSO was lower than expected because it failed to converge to the desired results within a limit of 1,000,000 iterations. Consider, for instance, Schwefel's Problem 2.21 (F2): for 30 runs, on 24 occasions, the standard PSO failed to reach the expected results. This means that the standard PSO is more susceptible to stagnate at local optima.

However, it is possible to observe that all models occasionally present convergence problems. Griewank's function (F5) is a function that easily leads the algorithms to local optima. There were instances in all topologies (except the broadcast) that did not converge to the desired results. It is important to highlight that the asynchronous communication mechanism helped the broadcast model to converge in all rounds. As the social component was not changed at a fixed period, this mechanism was more successful in avoiding local optima. The downside of the broadcast mechanism is the high cost of communication, as a large number of messages is employed.

For Rosenbrock's function (F3), a similar behavior was observed for the gossip to one-neighbor strategy. This communication strategy not only employed the smallest number of messages but also presented approximately the same number of iterations when compared with other methods. For Rastrigin's function (F4), the broadcast topology employed less iterations, although at a greater communication cost than the dynamic topology. Figure 4 shows the amount of communication in terms of the number of senders and messages sent for Schwefel's Problem 2.21 (F2), on the dynamic topology (left). The figure shows results of one execution. The horizontal x-axis represents the number of iterations (time) and the vertical y-axis represents the amount of communication performed. As mentioned before, a communication occurs when a swarm improves its result and sends it to other swarms.

In Figure 4 (left), we observe that less than 25% of the messages are transmitted during the first half of the execution (77501 iterations). It is possible to see that as the connectivity decreased, the number of messages exchanged also decreased, and there was also an improvement on the time required to obtain the results.

To illustrate situations that did not converge, we show two runs on the dynamic topology. The function chosen for this purpose was Shifted Griewank's (F5). In Figure 4 (right), the black curve represents executions that converged and the gray curve represents those that did not converge. We observe that the algorithm must escape from local optima in the first few iterations. When the function gets stuck in a local optimum, the number of iterations does not help reduce the convergence time. As shown by Figure 4 (right, gray line), there was no improvement of the results even when the number of iterations soared to between 160,000 and 1,000,000 iterations.

DOI: 10.1002/cpe

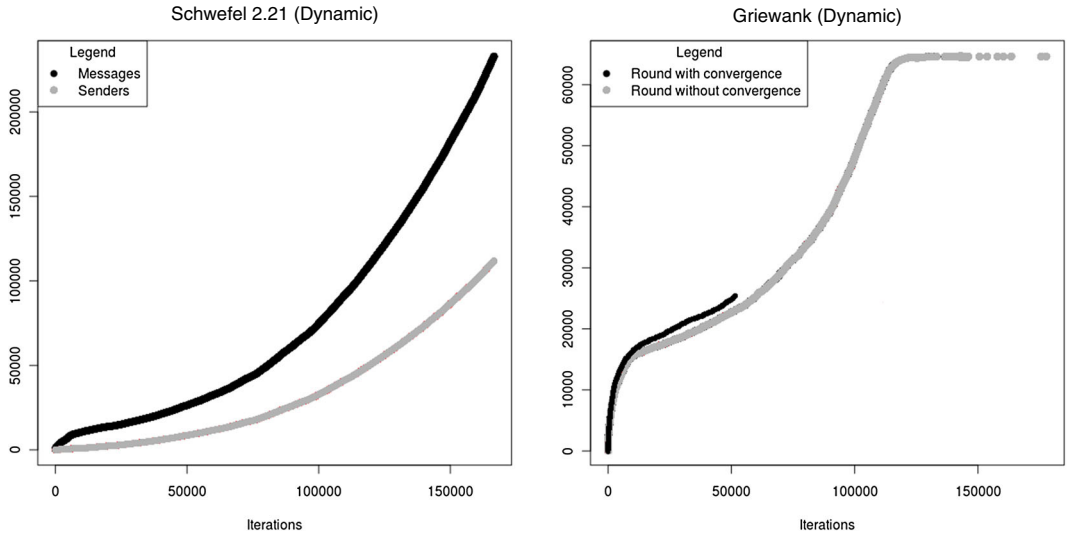

Figure 4. Communication overhead for (left) Schwefel's 2.21 (F2) and (right) Griewank's (F5) functions on the dynamic topology.

A characteristic we observed on all topologies and all functions was that at the beginning of the optimization, the *gbest* is easily improved and thus a large number of messages is exchanged among the swarms. However, after the first iterations, these improvements on the *gbest* occur less frequently and less messages are exchanged.

In the synchronous communication model, as communication always takes place at a fixed interval, a swarm may share poor *gbest* values unnecessarily. The asynchronous communication model does seem to improve the communication overhead, as swarms only send a message when there is a real improvement on the *gbest*. Consider for instance the Sphere function (F1). In most of multi-swarm approaches, one message is exchanged per iteration. In the gossip to one-neighbor topology, an average of 47,358.03 iterations were needed to reach the desired result, and in average, 6965.20 message exchanges were performed. In other words, only one communication message was exchanged for about 6.8 iterations.

We can conclude that a low connectivity exerted a positive influence on the results. For example, consider Rosenbrock's function (F3). Figure 5 depicts the box-and-whisker diagram (boxplot) of results. As noted, besides helping save messages (right), the gossip to one-neighbor topology showed the best results (left) of all topologies.

To evaluate the individual performance of each topology, we applied the evaluation criteria used in [23]. To sort the topologies from best to worst, the *Success Performance* for each problem was evaluated.

The success rate for each topology $= mean(\text{successful runs}) * \frac{(\#of totalruns)}{(\#of successfulruns)}$. The expressions are computed for each problem separately. Next, the Success Performance of each topology is divided by the Success Performance of the best algorithm for each function. Thus, the best topology for each function will have the value 1. As this value increases, it defines a ranking, from the best to the worst topology. With the Success Performance, we can sort the topologies. Table III shows this ranking.

When we applied the Friedman test on the Success Performance results, it accused no differences among the topologies. For example, the broadcast topology was the best for F1 (Sphere function) and the worst for F2 (Schwefel's 2.21 Problem). In other words: the best topology for a given function can also be the worst for another function. Thus, it is recommended to evaluate which topology is the best for a specific function. From these experiments, we can reach the conclusion that the execution of multiple swarms using several different topologies presented a positive effect on the convergence of the optimization process. This strategy allowed an improved exploration of the state space while restricting communication events that are only triggered by actual improvements.

DOI: 10.1002/cpe

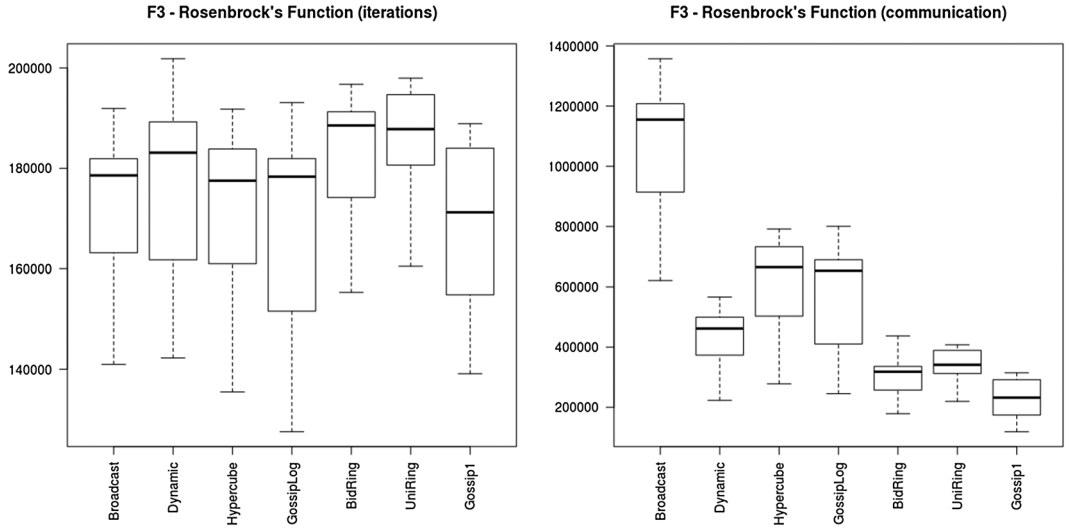

Figure 5. Box-and-whisker plot for Rosenbrock's function (F3): (left) convergence and (right) communication.

Table III. Ranking of topologies based on *Success Performance*.

Topology	F1	F2	F3	F4	F5	F6
Broadcast	1st	7th	4th	1st	1st	2nd
Dynamic	5th	2nd	6th	5th	4th	7th
Hypercube	3th	6th	2th	2th	3th	3th
Gossip log(n)	2th	5th	3th	3th	2nd	1st
Bidirectional ring	6th	3th	5th	6th	5th	5th
Unidirectional ring	7th	4th	7th	7th	7th	6th
Gossip 1	4th	1st	1st	4th	6th	4th

The broadcast and gossip topologies presented the best results in terms of the convergence. Another conclusion is that it is a good strategy to begin with a highly connected topology and later decrease the connectivity, as improving the social component becomes a rarer event as the iterations progress. However, when and how to decrease the connectivity still remains an open question, and may depend on several factors, such as the function being optimized.

Influence of the number of islands

We also investigated the performance impact of changing the number of islands in the optimization performance. We executed experiments as follows. Initially two functions were chosen to be optimized: one considered 'easy' (Sphere function - F1) and another considered 'difficult' (Rosenbrock's function - F3). Such functions were executed with 8, 16 and 32 swarms, and for each function, 30 runs were executed on each topology. The best result and the cost in terms of communication for each execution were stored after 60,000 iterations.

Table IV shows the mean and the standard deviation for each function and each topology with 8, 16, and 32 swarms.

To compare the results, the Kruskal–Wallis test was applied with significance level equal to 5%. The R toolkit for statistical computing was employed [26]. For the instances that had a significant statistical difference according to the test, the best averages are highlighted in bold. The results for each topology were analyzed according to the number of swarms. The test showed that, for both functions, the configuration with eight islands presented the best results. Thus, it is possible to conclude that keeping a few swarms with many particles was a good strategy in most of cases.

DOI: 10.1002/cpe

Table IV. Multi-warm performance with 8, 16 and 32 islands for Sphere and Rosenbrock's functions.

Topology		F1			F3		
		8 Nodes	16 Nodes	32 Nodes	8 Nodes	16 Nodes	32 Nodes
Broadcast	Average	**0.000004**	0.000004	0.000005	**186.97**	208.09	231.12
	Std. Dev.	**0.000001**	0.000001	0.000001	**99.938**	166.75	239.77
Dynamic	Average	**0.000013**	0.000053	0.000337	**263.35**	386.79	449.06
	Std. Dev.	**0.000003**	0.000011	0.000061	**209.47**	397.68	387.35
Hypercube	Average	**0.000002**	0.000005	0.000015	**163.93**	334.40	323.04
	Std. Dev.	**0.000009**	0.000020	0.000052	**212.36**	282.14	254.82
Gossip	Average	**0.000063**	0.000012	0.000005	**189.83**	297.76	284.83
Log(N)	Std. Dev.	**0.000013**	0.000003	0.000001	**55.25**	386.98	297.25
Bidirectional	Average	**0.000024**	0.000122	0.000908	**307.45**	360.17	237.50
Ring	Std. Dev.	**0.000005**	0.000028	0.000146	**333.66**	396.21	197.69
Unidirectional	Average	**0.000036**	0.000205	0.003033	**253.84**	360.61	398.19
Ring	Std. Dev.	**0.000008**	0.000031	0.000422	**227.78**	405.52	353.20
Gossip 1	Average	**0.000020**	0.000072	0.000305	**222.03**	212.79	495.26
	Std. Dev.	**0.000006**	0.000013	0.000072	**175.21**	222.35	432.73

Note: For the instances that had a significant statistical difference according to the test, the best averages are highlighted in bold.

However, it is important to note that, just as there is no single topology that presents the best results for all functions, there is not a single number of swarm that is suitable for all functions.

5. CONCLUSION

In this work, we investigated the impact of the topology in solving complex problems using asynchronous MSPSO. Several different topologies/communication strategies were considered: fully connected, broadcast, gossip with two different fanouts, unidirectional ring, bidirectional ring, hypercube, and a dynamic topology. Six benchmark functions were evaluated, each with 100 dimensions. Extensive experimental results are presented.

From the observation of experimental results, it is possible to conclude that the parallel execution of multiple swarms has a positive effect on the convergence of the optimization process by improving the search state exploration as separate populations are maintained. It can be observed that for simple functions, a high connectivity generally improves the convergence. However, for complex functions, a high connectivity may lead results to local optima. In some cases, topologies with lower connectivity were faster and employed less communication. The dynamic topology that starts with high connectivity, which is progressively reduced, proved to be a good strategy in several cases. This is particularly so because of the adopted asynchronous communication strategy. In the beginning, the *gbest* improved continually, generating a large number of messages. However, over time, the number of messages is reduced as the *gbest* improves less frequently.

Considering the individual performance of each topology, a ranking was defined. When we applied a statistical test on the results, it accused no differences among the topologies. That is, the best topology for a given function can also be the worst for another function.

We also investigated the impact of varying the number of islands in the optimization performance. Simple and complex functions were evaluated and for both types of functions, the configuration with eight islands presented the best results. In other words, keeping a few swarms with many particles proved to be a good strategy in most of the cases.

Future work includes expanding the evaluation to consider functions with different behaviors in order to determine the best alternatives for each kind of problem. We also plan to focus on multi-objective optimization problems. New metrics to evaluate the convergence and diversity can be used, such as the spread, coverage, hypervolume, and generational distance, among others. Other types of communication frameworks for running parallel and distributed optimization, such as peer-to-peer networks, parallel computing platforms including multicore processors [7, 12].

DOI: 10.1002/cpe

ACKNOWLEDGEMENTS

This work was partially supported by the Brazilian National Research Council (CNPq), projects 304013/2009-9 and 303761/2009-1, and Araucaria Foundation (Fundação Araucária), project 14737.

REFERENCES

1. Kennedy J, Eberhart R. Particle swarm optimization. In *Neural Networks, 1995. Proceedings., IEEE International Conference on*, Vol. 4. Institute of Electrical & Electronics Enginee: Perth, Australia, 1995; 1942–1948.
2. Zheng X, Chen K, Lu D, Liu H. A proposal for a cooperative coevolutionary PSO. *Information Technologies and Applications in Education, 2007. ISITAE '07. First IEEE International Symposium on*, Kunming, China, 2007; 324 –329.
3. Geng Z, Zhu Q. Multi-swarm PSO and its application in operational optimization of ethylene cracking furnace. *Intelligent Control and Automation, 2008. WCICA 2008. 7th World Congress on*, Chongqing, China, 2008; 103 –106.
4. Xiangwei Z, Hong L. A different topology multi-swarm PSO in dynamic environment. *IT in Medicine Education, 2009. ITIME '09. IEEE International Symposium on,* Vol. 1, Hong, L, Xiangwei, Z. (eds). Ji'nan, China, 2009; 790 –795.
5. Chakraborty P, Das S, Abraham A, Snasel V, Roy GG. On convergence of multi-objective particle swarm optimizers. *Evolutionary Computation (CEC), 2010 IEEE Congress on*, Barcelona, Spain, 2010; 1–8.
6. Li Y, Liang J, Hu J. A multi-swarm cooperative hybrid particle swarm optimizer. *Natural Computation (ICNC), 2010 Sixth International Conference on,* Vol. 5, Yue, S, Wei, H -L, Wang, L, Song, Y. (eds). Yantai, Shandong, China, 2010; 2535–2539.
7. de Vega FF, Cantú-Paz E (eds). *Parallel and Distributed Computational Intelligence*, Vol. 269. Springer: Springer Berlin / Heidelberg, 2010.
8. El-Abd M, Kamel MS. A taxonomy of cooperative particle swarm optimizers. *International Journal of Computational Intelligence Research* 2008; **4**:137–144.
9. Montes de Oca M, Stutzle T, Birattari M, Dorigo M. Frankenstein's PSO: a composite particle swarm optimization algorithm. *Evolutionary Computation, IEEE Transactions on* 2009; **13**:1120 –1132.
10. Kennedy J. Small worlds and mega-minds: effects of neighborhood topology on particle swarm performance. *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*, Vol. 3, Washington DC, USA, 1999; 1931–1938.
11. Janson S, Middendorf M. A hierarchical particle swarm optimizer and its adaptive variant. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on* 2005; **35**:1272 –1282.
12. Tomassini M, Vanneschi L. Introduction: special issue on parallel and distributed evolutionary algorithms, part i. *Genetic Programming and Evolvable Machines* 2009; **10**:339–341.
13. Alba E, Troya JM. A survey of parallel distributed genetic algorithms. *Complexity* 1999; **4**:31–52.
14. van den Bergh F, Engelbrecht AP. A cooperative approach to particle swarm optimization. *Evolutionary Computation, IEEE Transactions on* 2004; **8**:225 –239.
15. Li X, Yao X. Tackling high dimensional nonseparable optimization problems by cooperatively coevolving particle swarms. *Evolutionary Computation, 2009. CEC '09. IEEE Congress on*, Trondheim, Norway, 2009; 1546 –1553.
16. Li X, Yao X. Cooperatively coevolving particle swarms for large scale optimization. *Evolutionary Computation, IEEE Transactions on* 2012; **16**(2):210–224.
17. Liang J, Suganthan P. Dynamic multi-swarm particle swarm optimizer. *Swarm Intelligence Symposium, 2005. SIS 2005. Proceedings 2005 IEEE*, Pasadena, California, 2005; 124–129.
18. Vanneschi L, Codecasa D, Mauri G. A study of parallel and distributed particle swarm optimization methods. In *Proceeding of the 2nd Workshop on Bio-Inspired Algorithms for Distributed Systems*. ACM: ACM New York, NY, USA, 2010; 9–16.
19. Potter MA, Jong KAD. A cooperative coevolutionary approach to function optimization. In *Parallel Problem Solving from Nature*. Springer-Verlag: Springer-Verlag, Berlin, 1994; 249–257.
20. Lvbjerg M, Rasmussen TK, Krink T. Hybrid particle swarm optimiser with breeding and subpopulations. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001*. Morgan Kaufmann: San Francisco, California, USA, 2001; 469–476.
21. Renesse RV, Minsky Y, Hayden M. A gossip-style failure detection service, 1998; 55–70.
22. Tang K, Yao X, Suganthan PN, MacNish C, Chen YP, Chen CM, Yang Z. *Benchmark Functions for the CEC'2008 Special Session and Competition on Large Scale Global Optimization*, Nature Inspired Computation and Applications Laboratory, 2007. (Available from: http://nical.ustc.edu.cn/cec08ss.php) [Accessed May 2011].
23. Suganthan PN, Hansen N, Liang JJ, Deb K, Chen YP, Auger A, Tiwari S. Problem definitions and evaluation criteria for the CEC 2005 special session on real-parameter optimization Nanyang Technological University. *Natural Computing* 2005; (May):1–50. http://vg.perso.eisti.fr/These/Papiers/Bibli2/CEC05.pdf.
24. MacDougall MH. *Simulating Computer Systems: Techniques and Tools*. MIT Press: Cambridge, Massachusetts, 1987.
25. Eberhart RC, Shi Y. Comparing inertia weights and constriction factors in particle swarm optimization. *Evolutionary Computation, 2000. Proceedings of the 2000 Congress on*, Vol. 1, San Diego, CA, USA, 2000; 84–88.
26. R project for statistical computing, 2011. (Available from: http://www.r-project.org/) [Accessed December 2011].

DOI: 10.1002/cpe