Swarm and Evolutionary Computation 18 (2014) 22–37

Contents lists available at ScienceDirect

Swarm and Evolutionary Computation

journal homepage: www.elsevier.com/locate/swevo

Regular Paper

A hybrid particle swarm with a time-adaptive topology for constrained optimization

Mohammad Reza Bonyadi*, Xiang Li, Zbigniew Michalewicz

School of Computer Science, The University of Adelaide, Australia

ARTICLE INFO

Article history:
Received 25 January 2014
Received in revised form
3 May 2014
Accepted 7 June 2014
Available online 16 June 2014

Keywords:
Particle swarm optimization
Continuous space optimization
Constrained optimization problems
Disjoint feasible regions

ABSTRACT

For constrained optimization problems set in a continuous space, feasible regions might be disjointed and the optimal solution might be in any of these regions. Thus, locating these feasible regions (ideally all of them) as well as identifying the most promising region (in terms of objective value) at the end of the optimization process would be of a great significance. In this paper a time-adaptive topology is proposed that enables a variant of the particle swarm optimization (PSO) to locate many feasible regions at the early stages of the optimization process and to identify the most promising one at the latter stages of the optimization process. This PSO variant is combined with two local searches which improve the ability of the algorithm in both finding feasible regions and higher quality solutions. This method is further hybridized with covariance matrix adaptation evolutionary strategy (CMA-ES) to enhance its ability to improve the solutions at the latter stages of the optimization process. Results generated by this hybrid method are compared with the results of several other state-of-the-art methods in dealing with standard benchmark constraint optimization problems.

1. Introduction

A constrained optimization problem (COP) in a continuous space is formulated as follows:

$$\text{Find } x \in S \subseteq R^D \text{ such that} \begin{cases} \forall y \in \mathcal{F}\; f(x) \le f(y) & (a) \\ g_i(x) \le 0, \quad \text{for } i=1 \text{ to } q & (b) \\ h_i(x) = 0, \quad \text{for } i=1 \text{ to } p & (c) \end{cases} \tag{1}$$

In this formulation, f, g_i, and h_i are real-valued functions defined on the search space S, q is the number of inequalities, and p is the number of equalities. The search space S is defined as a D dimensional rectangle in R^D such that $l(j) \le x(j) \le u(j)$, $j=1,\ldots, D$ ($l(j)$ and $u(j)$ are the lower and upper bounds of the jth variable). The set of all feasible points which satisfy constraints (b) and (c) are denoted by $\mathcal{F}$ [1]. Usually in COPs, equalities are replaced by inequalities [2] as follows:

$$|h_i(x)| \le \xi, \quad \text{for } i=1 \text{ to } p \tag{2}$$

where ξ is a small positive value. In all experiments reported in this paper, the value of ξ is equal to $1E-4$, the same as it was adopted in [2,3]. Accordingly, by considering $g_{i+q}(x) = |h_i(x)| - \xi$ for all $1 \le i \le p$, the COP defined in Eq. 1 can be written as

$$\text{Find } x \in S \subseteq R^D \text{ such that} \begin{cases} \forall y \in \mathcal{F}\; f(x) \le f(y) & (a) \\ g_i(x) \le 0, \quad \text{for } i=1 \text{ to } q+p & (b) \end{cases} \tag{3}$$

From now on, the term COP refers to this formulation.

Any method that deals with a COP consists of two parts: an optimization algorithm and a constraint handling technique (CHT). The optimization algorithm can be particle swarm optimization (PSO) [4], genetic algorithm (GA) [5], differential evolution (DE) [6], covariance matrix adaptation evolutionary strategy (CMA-ES) [7], conjugate gradient [8], linear programming [9], etc. However, whatever the optimization algorithm is, evaluation of individuals is one of the challenges in solving COPs [10]. Indeed, unlike unconstrained optimization problems in where evaluation is simply done based on the value of the objective function for each individual, evaluation procedure for COPs includes some complexities because it is necessary to consider both constraints and objective value (see [11] for detailed discussion on this complexity). There are several categories of techniques in handling constraints that can be incorporated into optimization algorithms [12]; these categories include: penalty functions, special operators, repairs, decoders, and hybrid techniques (see also [1] and [10] for details).

Particle Swarm Optimization (PSO) [13] is a population based optimization algorithm of $n > 1$ particles (referred to as *swarm*); each particle is defined by three D-dimensional vectors

* Corresponding author.
E-mail addresses: mbonyadi@cs.adelaide.edu.au, vardiar@gmail.com (M. Reza Bonyadi), xiang.li01@cs.adelaide.edu.au (X. Li), zbyszek@cs.adelaide.edu.au (Z. Michalewicz).

http://dx.doi.org/10.1016/j.swevo.2014.06.001
2210-6502/© 2014 Elsevier B.V. All rights reserved.

- *Position* ($\vec{x}^{i}_{t}$) – is the position of the ith particle in the tth iteration. This is used to evaluate the particle's quality.
- *Velocity* ($\vec{v}^{i}_{t}$) – is the direction and length of movement of the ith particle in the tth iteration.
- *Personal best* ($\vec{p}^{i}_{t}$) – is the best position[1] that the ith particle has visited in its lifetime (up to the tth iteration). This vector serves as a memory for keeping knowledge of quality solutions [4].

All of these vectors are updated at every iteration t for each particle (i)

$$\vec{v}^{i}_{t+1} = \mu(\vec{x}^{i}_{t}, \vec{v}t^{i}, N^{i}_{t}) \text{ for all } i \tag{4}$$

$$\vec{x}^{i}_{t+1} = \xi(\vec{x}^{i}_{t}, \vec{v}^{i}_{t+1}) \text{ for all } i \tag{5}$$

In Eq. 4, N^i_t (known as neighbor set of the particle i at iteration t) is a subset of personal best positions of some particles which contribute to the velocity updating rule of that particle at iteration t, i.e. $N^i_t = \{\vec{p}^{k}_{t} | k \in \{T^i_t \subseteq \{1, 2, \ldots, n\}\}\}$ where T^i_t is a set of indices of particles which contribute to the velocity updating for particle i at iteration t. Clearly, the strategy of determining T^i_t might be different for various types of PSO algorithms and it is usually referred to as the *topology* of the swarm. Many different topologies have been defined so far [14], e.g., global best topology (gbest), ring topology, non-overlapping, pyramid, and adaptive topology, that are discussed later in this paper. The function $\mu(.)$ calculates the new velocity vector for particle i according to its current position, current velocity $\vec{v}^{i}_{t}$, and neighborhood set N^i_t. In Eq. 5, $\xi(.)$ is a function which calculates the new position of the particle i according to its previous position and its new velocity. Usually $\xi(\vec{x}^{i}_{t}, \vec{v}^{i}_{t+1}) = \vec{x}^{i}_{t} + \vec{v}^{i}_{t+1}$ is accepted for updating the position of particle i. After updating velocity and positions, the personal best vector ($\vec{p}^{i}_{t}$) of the particles is also updated.

$$\vec{p}^{i}_{t+1} = \begin{cases} \vec{p}^{i}_{t} & f(\vec{p}^{i}_{t}) \le f(\vec{x}^{i}_{t+1}) \\ \vec{x}^{i}_{t+1} & \text{otherwise} \end{cases} \tag{6}$$

In Eq. (6), the new personal best position for the ith particle is updated according to the objective values of its previous personal best and the current position. In the rest of this paper, these usual forms for the position updating rule (Eq. (5)) and for updating the personal best (Eq. (6)) are assumed. In PSO, updating rules (Eqs. (4) and (5)) are applied to all particles and the personal best for all particles are updated in each iteration until a predefined termination criterion, e.g., maximum number of iterations or deviation from global optimum (if known), is met.

In the original version of PSO [13], the function $\mu(\cdot)$ in Eq. (4) was defined as

$$\vec{v}^{i}_{t+1} = \vec{v}^{i}_{t} + \varphi_1 R^i_{1t} \underbrace{(\vec{p}^{i}_{t} - \vec{x}^{i}_{t})}_{\substack{\text{Personal} \\ \text{Influence (PI)}}} + \varphi_2 R^i_{2t} \underbrace{(\vec{g}_{t} - \vec{x}^{i}_{t})}_{\substack{\text{Social} \\ \text{Influence (SI)}}} \tag{7}$$

where φ_1 and φ_2 are two real numbers known as acceleration coefficients[2] and $\vec{p}^{i}_{t}$ and $\vec{g}_{t}$ are the personal best (of particle i) and the global best vectors, respectively, at iteration t. Also, the role of vectors $PI = \vec{p}^{i}_{t} - \vec{x}^{i}_{t}$ (Personal Influence) and $SI = \vec{g}_{t} - \vec{x}^{i}_{t}$ (Social Influence) is to *attract* the particles to move toward known quality solutions, i.e. personal and global best. Moreover, R_{1t} and R_{2t} are two $d \times d$ diagonal matrices[3] [15,16], where their elements are random numbers distributed uniformly ($\sim U(0, 1)$) in $[0, 1]$. Note that matrices R_{1t} and R_{2t} are generated at each iteration for each particle separately.

In 1998, Shi and Eberhart [17] introduced a new coefficient ω (known as *inertia weight*) to control the influence of the previous velocity value on the updated velocity. Indeed, Eq. 7 was written as

$$\vec{v}^{i}_{t+1} = \omega \vec{v}^{i}_{t} + \varphi_1 R^i_{1t}(\vec{p}^{i}_{t} - \vec{x}^{i}_{t}) + \varphi_2 R^i_{2t}(\vec{g}_{t} - \vec{x}^{i}_{t}) \tag{8}$$

The coefficient ω controls the influence of the previous velocity ($\vec{v}^{i}_{t}$) on the movement of the particle (this variant is called Standard PSO, SPSO, throughout this paper). One of the issues in SPSO was that, for some values of the coefficients, velocity may grow to infinity. Some studies analyzed the dynamic of the particles to understand why velocity might grow to infinity. It was proven that by setting the coefficients in specific boundaries, velocity shrinks during the time and hence, it does not grow to infinity [18–20]. In SPSO, if the random matrices are replaced by random values then the new variant is known as linear PSO (LPSO).

There are several well-studied issues in the standard PSO such as stagnation [21–24], line search [25,26], and swarm size [21,22]. Apart from these issues in PSO, there have been some attempts to extend the algorithm to work with COPs [3,27–38] and to support niching[4] [39–42]. See Section 2 for a brief review on the issues and extensions of SPSO.

In this paper, different topologies for a PSO variant proposed in our earlier paper [11] are analyzed and their abilities in locating disjoint feasible regions of a COP are tested. Consequently, this variant is extended by a new time-adaptive topology which enables the algorithm to locate feasible regions at the early stages of iterations and to find the region with the highest quality (in terms of the objective function) at the latter stages of the optimization process. Also, this extended method is combined further with two local searches and a covariance matrix adaptation evolutionary strategy (CMA-ES) [43] to improve the quality of the found solutions. The hybrid approach is applied to standard benchmark COPs (usually known as CEC2010 [44]) and its results are compared with three other recently proposed approaches [2,45,46].

The rest of this paper is organized as follows. Section 2 provides an overview of PSO including discussion on some identified issues of this technique, its topology, niching capabilities, and its applicability for COPs. Section 3 discusses two constraint handling methods as well as some relevant optimization methods to deal with COPs. In Section 4 a PSO variant is extended by a new time adaptive topology and the extended method is combined with local searches. Experimental results are reported and analyzed in Section 5 and Section 6 concludes the paper.

2. Particle swarm optimization

In this section we provide an overview of PSO, including issues in the algorithm, topology, niching abilities, and its ability to deal with COPs.

[1] In general, personal best can be a *set* of best positions, but all PSO types listed in this paper use single personal best.

[2] These two coefficients control the effect of personal and global best vectors on the movement of particles and they play an important role in the convergence of the algorithm. They are usually determined by a practitioner or by the dynamic of particles' movement.

[3] Alternatively, these two random matrices are often considered as two random vectors. In this case, the multiplication of these random vectors by *PI* and *SI* is element-wise.

[4] Niching is the ability of the algorithm to locate different optima rather than only one of them. The niching concept is used usually in the multi-modal optimization.

2.1. Some issues in PSO

In SPSO, for some values of acceleration coefficients and inertia weight, the velocity vector might grow to infinity (the issue is called *swarm explosion*). Swarm explosion results in moving particles to infinity which is not desirable [19]. One of the early solutions for this issue was to restrict the value of each dimension of the velocity in a particular interval $[-V_{max}, V_{max}]$ where V_{max} can be considered as the maximum value of the lower bound and upper bound of the search space [47] (this is known as *nearest* strategy). Also, there are some other strategies to restrict the velocity in a way that the swarm explosion is prevented (e.g., nearest with turbulence, random). However, none of these strategies is comprehensive enough to prevent the swarm explosion effectively in general (see [48] for details). In fact, these strategies limit the step sizes for updating the particles positions, while particles still may move to infinity. Thus, many researchers analyzed the behavior of the particles to find the reasons behind swarm explosion from different point of views [19,20,49]. The aim of these analyses was to define criteria for the acceleration coefficients and inertia weight such that particles converge to a point, which actually prevent the velocity from growing to infinity. One of the earliest attempts in this sort was made in [19] where a constriction coefficient PSO (CCPSO) was proposed. The authors revised the velocity updating rule to

$$\vec{V}^i_{t+1} = \chi(\vec{V}^i_t + c_1 r_{1t}(\vec{p}^i_t - \vec{x}^i_t) + c_2 r_{2t}(\vec{g}_t - \vec{x}^i_t)) \tag{9}$$

In this equation, χ is called the *constriction factor* and it is proposed to set its value by

$$\chi = 2k/|2 - c - \sqrt{c^2 - 4c}| \tag{10}$$

where $c = c_1 + c_2 > 4$. The authors proved that if these conditions hold for the constriction factor, particles converge to a stable point and the velocity vector does not grow to infinity. The values of c_1 and c_2 are often set to 2.05 and the value of k is set to 1. In [50], it was proven that for any c_1 and c_2 which satisfy converging conditions, all particles collapse on the global best vector of the swarm ($\vec{g}_t$), i.e. $\lim_{t\to\infty} \vec{x}^i_t = \vec{p}^i_t = \vec{g}_t$ for all particles, with probability 1. Also, for all particles, the velocity vector shrinks to zero. Thus, in this situation ($\vec{g}_t = \vec{p}^i_t = \vec{x}^i_t$ for all particles and at the same time $v\vec{\ }^i_t = 0$), all particles stop moving and no improvement can take place as all components for moving the particles are zero. This analysis was also done from other perspectives by [20,49,51].

Although constriction coefficient guarantees converging particles to a point (a convergent sequence), there is no guarantee that this final point is a quality point in the search space [49]. In fact, the point $\vec{g}_t$ might not be a local optimum of the search space, which means that there is no guarantee that the algorithm can locate a local optimum. This issue is called *stagnation* in this paper (converging to a point that is not a local optimum) that was first identified as a defect in SPSO [21] and further investigated by [22,52,53]. This issue exists in both LPSO and SPSO.

Another issue that is exclusive to LPSO is called *line search* [25,45]: if $\vec{g}_t$, $\vec{p}^i_t$, and $\vec{x}^i_t$ are on the same line and $v\vec{\ }^i_t$ is in parallel with $(\vec{p}^i_t - \vec{x}^i_t)$ or $(\vec{g}_t - \vec{x}^i_t)$ (i.e. $((\vec{p}^i_t - \vec{x}^i_t) || (\vec{g}_t - \vec{x}^i_t)$ and $\vec{V}^i_t || (\vec{p}^i_t - \vec{x}^i_t))$), the particle i starts oscillating on the line segment connecting its personal best and the global best (line search). In this case, only the points that are on this line are sampled by the particle i and other locations in the search space are not examined anymore. It was shown [25] that this is not the case in SPSO, however, there are some situations that the particles in standard PSO start oscillating along one of the dimensions while there is no chance for them to get out of this situation [22,26].

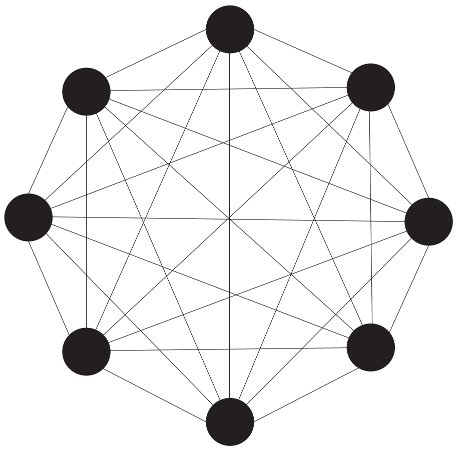

Fig. 1. A swarm with the gbest topology. Each circle represents one particle.

Stagnation happens with higher probability when swarm size is small [21] (this is called *swarm size* issue throughout the paper). In [21], it was argued that SPSO is not effective when its swarm size is small (2 for example) and particles stop moving in the earlier stage of the optimization process. To address this issue, a new velocity updating rule was proposed that was only applied to the global best particle to prevent it from becoming zero (this variant was called Guaranteed Convergence PSO, GCPSO). Consequently, the global best particle never stops moving which solves the stagnation issue and, as a result, swarm size issue is addressed as well. Experiments confirmed that, especially in the single modal optimization problems, the new algorithm is significantly better than the standard version when the swarm size is small (with 2 particles). Note that, in LPSO, apart from stagnation issue, the line search issue can be another reason that the algorithm becomes ineffective when swam size is small.

2.2. Topology in PSO

Many different topologies have been introduced so far for PSO [54]. One of the well-known topologies is called *gbest* topology. In this topology, the set T^i_t contains all particles in the swarm, i.e. $T^i_t = \{1, 2, \ldots, n\}$. As an example, SPSO uses this topology as in each iteration, $\vec{g}_t$ is used for the velocity updating rule and $\vec{g}_t = \vec{p}^{\tau_t}_t$ where $\tau_t = \underset{l \in T^i_t}{argmin}\{F(\vec{p}^l_t)\}$. Fig. 1 shows this topology.

It was shown that when this topology is used, the algorithm converges rapidly to a point [54]. The reason is that all particles are connected[5] to each other, hence, they all tend to converge to the best ever found solution.

Another well-known topology is called *ring* topology, where the set T^i_t contains $\{i, i-1, i+1\}$ (it is assumed that the particles are in a fixed order during the run). In fact, each particle is connected to two other particles that are the previous and the next particles. Also, if $i+1$ was bigger than n (swarm size), it is replaced by 1, and if $i-1 < 1$, it is replaced by n. Fig. 2 shows this topology.

The velocity updating rule for this topology is written as

$$\vec{V}^i_{t+1} = \vec{V}^i_t + \varphi_1 R^i_{1t}\underbrace{(\vec{p}^i_t - \vec{x}^i_t)}_{\substack{\text{Personal}\\ \text{Influence (PI)}}} + \varphi_2 R^i_{1t}\underbrace{(\vec{lb}^i_t - \vec{x}^i_t)}_{\substack{\text{Social}\\ \text{Influence (SI)}}} \tag{11}$$

[5] A particle i is *connected* to particle j if it is aware of the personal best location of the particle j.

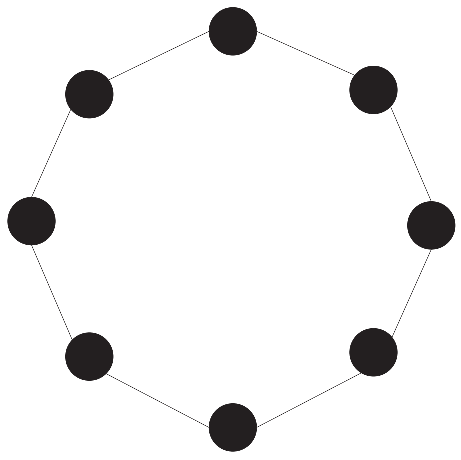
Fig. 2. A swarm with the ring topology.

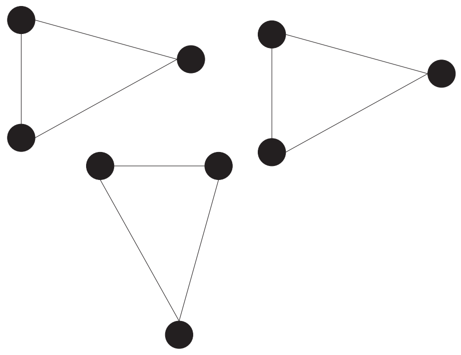
Fig. 3. A swarm with the non-overlapping topology with three particles in each sub-swarm.

where $\vec{lb}_t^i$ is the best ever found solution by the particles i, $i-1$, and $i+1$, i.e. $\vec{lb}_t^i = p_{\vec{t}}^{\tau_t^i}$ where $\tau_t^i = \underset{l \in T_t^i}{argmin}\{F(\vec{p}_t^l)\}$. It was shown that this topology causes the algorithm to spend more iterations for exploration (compared to gbest topology), which results in better explorative behavior [55].

Another topology that is used in this paper is called *non-overlapping* topology [56]. In this topology, all particles in the swarm are divided into several sets (called sub-swarms) that are independent from each other. In fact, we define the set $T_t^i = \{\{i\} \cup \{s \subseteq \{1, 2, \ldots, n\}\}\}$, which refers to all connected particles to particle i. In any non-overlapping topology at every iteration, there exist at least one particle i that for all j as a member of $\{\{1,2,\ldots,n\} - T_t^i\}$, the intersection of T_t^i and T_t^j is empty, i.e. $\forall t\ \exists i \in \{1,2,\ldots n\}\ \forall j \in \{\{1,2,\ldots,n\} - T_t^i\}\ \{T_t^i \cap T_t^j\} = \varnothing$. To update velocity for a particle i which uses this topology, Eq. 11 is used where $\vec{lb}_t^i$ is the best personal best among all particles in the set T_t^i, i.e. $\vec{lb}_t^i = p_{\vec{t}}^{\tau_t^i}$ where $\tau_t^i = \underset{l \in T_t^i}{argmin}(F(\vec{p}_t^l))$. Fig. 3 shows an example of this topology.

Note that, in this case, gbest topology is a special case of non-overlapping topology because for all i, the set $\{\{1,2,\ldots,n\} - T_t^i\}$ is empty and, consequently, T_t^j is also empty. This means that $\{T_t^i \cap T_t^j\} = \varnothing$ for any $j \in \{\{1,2,\ldots,n\} - T_t^i\}$. If the size of T_t^i is similar for all i, we represent the topology by the notation $n - v_l$ where l is the size of each sub-swarm. Thus, the gbest topology can be indicated by $n - v_n$. The best particle in each sub-swarm is defined as the particle that its personal best has the highest quality in that sub-swarm. This particle is called the *lead* particle in a sub-swarm while the rest of the particles in that sub-swarm are called *followers*. Note that the lead particle plays the rule of global best particle in gbest topology, i.e. the lead particle in each sub-swarm attracts followers in that sub-swarm.

There have been some adaptive topology approaches reported in literature. For example, in [57], the authors proposed a method called Club-based PSO (C-PSO) where the particles were members of clubs (each particle can be a member of one or more clubs) and each particle i communicated only with the particles in the club that it was in. A particle can be a member of several clubs, so that it communicates with all particles in all of those clubs. Note that the clubs can have overlap with each other. The membership degree of particle i is shown by $m(i)$ that represents the number of clubs that particle i is a member of. The value of $m(i)$ for all i was updated in each iteration according to the number of clubs the particle is a member of. The main idea of updating the membership degree was to reduce the number of clubs that high quality particles are in to prevent fast convergence. Also, the membership degree for the low quality particles is increased to give them a chance to learn from other particles in the swarm.

In [58], the authors experimented with two PSO variants with different topologies in two independent populations. One of these populations used global best topology (see Fig. 2) and the other used a new topology called *random tournament*. The velocity updating rule for the random tournament topology was the same as Eq. 11 where the vector $\vec{lb}_t$ was determined by using a tournament selection operator over β randomly selected particles. These two populations worked in parallel and at each iteration, the performance of these two populations were measured. According to this performance measure, particles are taken from one population and added to the other. The aim of this process was to make more use of the population that shows better performance. See [58] for further details.

The idea of "six degrees of separation" [59] (two arbitrary person are connected with their friends of the friends with maximum level of 6) was used to structure the connection between the particles in the swarm [60]. Every certain number of generations, k other particles are selected randomly for each dimension j of a particle i in the swarm. Then, the best particle among these k particles is chosen (particle h) and the value of the jth dimension for the particle i is updated by

$$\vec{v}_{t+1}^{ij} = \vec{v}_t^{ij} + \varphi_1 r_t^{ij}(\vec{p}_t^{hj} - \vec{x}_t^{ij}) \tag{12}$$

where r_t^{ij} is a random number uniformly distributed in the interval $[0, 1]$, and $p_{\vec{t}}^{kj}$ is the personal best of the hth particle. The number of generation to update the network of k particles was determined by an adaptive role which was based on the number of successive iterations that the particle has not been updated.

There are some other topologies (e.g., pyramid, wheel) and it is beyond the scope of this paper to review all of them. Our review has been limited to the topologies that we will use in the rest of the paper. Interested readers are referred to [14,54,55,61] for further information on various topologies.

2.3. Niching in PSO

Niching is a concept that has been introduced in multi-modal optimization. In multi-modal optimization, locating several (ideally all) optima (including local and global optima) by the algorithm is required. An optimization algorithm is said to support niching if it is able to locate different optima (also known as niches) in the search space rather than finding only one of them [62].

There have been many attempts to adopt PSO to support niching [40–42]. As an example, in [40], the authors analyzed the performance of PSO when gbest or ring topology is taken into account. In the gbest topology, results showed that only one optimum is located at each run of the algorithm. This was actually expected as all particles converge to $\vec{g}_t$ that does not support the niching aims. Also, the abilities of ring topology were investigated experimentally to understand whether ring topology can support niching aims. After applying PSO with the ring topology to some benchmark problems (5 test cases), the authors concluded [40] that the ring topology is not an appropriate candidate for niching.

A multi-swarm approach called NichePSO [41] was proposed in which multiple sub-swarms were run to locate different optimum solutions. Sub-swarms could merge together or exchange particles with one another. Also, in NichePSO, whenever the improvement in a particle's objective value over some number of iterations (a parameter) was small, a sub-swarm was created within that particle's neighbor to assist that particle in improving the solution.

As results reported in [40] for testing the ring topology were very limited, the ability of the ring topology for niching was reinvestigated in [42]. The author found that a PSO algorithm which uses the ring topology can operate as a niching algorithm because the personal best of each particle forms a stable network retaining the best positions found so far, while these particles explore the search space more broadly. Also, it was concluded that by using a reasonably large population, PSO algorithms which use the ring topology are able to locate dominant niches (optima) across the search space. This means that particles locate niches that are fairly similar in terms of their objective value. However, if the aim of the algorithm is to locate other local optima that are less dominant as well, a non-overlapping topology can be a good candidate. Results showed that a non-overlapping topology with 2 or 3 particles (i.e. $n-v_2$ or $n-v_3$) in each sub-swarm is significantly better than other topologies when the number of dimensions is small (up to 8 dimensions in the experiments conducted in [42]). However, their performance impaired much faster than other PSOs in locating optima as the number of dimensions grow. In fact, $n-v_2$ and $n-v_3$ were the worst methods among other tested methods when the number of dimensions was larger than 8, based on experiments.

2.4. PSO for COPs

There have been few attempts for enhancing PSO to handle COPs. The methods were based on penalty function [28,33], feasibility preservation [29], co-evolution [3], etc. In this subsection, some of the most recent approaches are reviewed.

In 2007, LPSO was applied [38] for solving COPs with linear equality constraints. In this approach, the random matrices in PSO (R_{1t} and R_{2t}) were replaced by random values (r_{1t} and r_{2t}). This modification enabled the algorithm to perform better search along the feasible space, in this case linear equalities. However, LPSO suffered from line search and stagnation issues [38]. To overcome the line search and stagnation in LPSO, the velocity updating rule for the global best vector was modified and another type of LPSO called converging LPSO (CLPSO) was proposed [38]. The global best particle in CLPSO was updated according to the equality constraints in the global best position. Experiments showed that the results of LPSO and CLPSO on some benchmark problems were comparable with that of Genocop II [63] (a GA-based optimization approach). Also, experiments indicated that for convex search spaces or the search space without many local optima, CLPSO worked considerably better than LPSO.

Cooperation comprehensive learning PSO (Co-CLPSO) was also implemented and applied to COPs [3]. In this method two sub-swarms cooperated with each other to solve the problem. Particles in each swarm were adaptively assigned to explore different constraints. Also, two swarms exchanged their information by regrouping the particles into different swarm. This enabled the particles to use the experiences of other particles in both swarms. Also, a local search based on SQP was proposed to improve the solutions during the run. This approach received the fourth place in the competition of CEC2010 in solving COPs [44].

In our earlier work, we investigated [45] the line search and stagnation issues for LPSO. It was shown that by applying a mutation operator to the velocity updating rule, both of these issues (line search and stagnation) are addressed (this variant is called Mutation Linear PSO, MLPSO). The velocity updating rule was revised to

$$\vec{v}^{\,i}_{t+1} = m(\omega\vec{v}^{\,i}_t + \varphi_1 r_{1t}(\vec{p}^{\,i}_t - \vec{x}^{\,i}_t) + \varphi_2 r_{2t}(\vec{g}_t - \vec{x}^{\,i}_t),\ c, \gamma) \tag{13}$$

where m is the mutation operator and c and γ are two constants. The operator m was defined as $m(\vec{d}, c, \gamma) = \vec{d} + N(0, \vec{\sigma})$ ($\vec{d} = \vec{v}^{\,i}_t + \varphi_1 r_{1t}(\vec{p}^{\,i}_t - \vec{x}^{\,i}_t) + \varphi_2 r_{2t}(\vec{g}_t - \vec{x}^{\,i}_t)$) and σ_j (the value of the jth dimension of $\vec{\sigma}$) was formulated as follows:

$$\text{for all } j \in \{1, \ldots, D\}\ \sigma_j = \begin{cases} c*\|\vec{d}\| & \text{if } \|\vec{d}\| > \gamma \\ c*\|N(0, \vec{\gamma})\| & \text{otherwise} \end{cases} \tag{14}$$

Indeed, the mutation operator used a multi-variant normal distribution with the mean equal to $d^{\rightarrow}$ and variance $c*\|d^{\rightarrow}\|$, where c is a constant. Also, in order to prevent the velocity from becoming zero, $\vec{d}$ was regenerate randomly when $\|\vec{d}\| \le \gamma$, where γ was a constant (set to $1E-10$). γ is actually a threshold which ensures velocity is non-zero. One important difference between MLPSO and GCPSO is that in GCPSO stagnation has been prevented *only* for the global best particle while stagnation in MLPSO has been prevented for all particles. MLPSO was extended by a constraint handling method called ε-level constraint handling (see Section 3 for details) to deal with COPs (this variant was called Epsilon-level Mutation Linear PSO, EMLPSO). The method was further extended by combining with CMA-ES (this variant was called EPSO-CMA). In fact, the best solution found by EMLPSO was used to initialize CMA-ES for further improvement. Results showed that the hybrid method is effective in dealing with COPs.

3. Some constraint handling techniques

In this section two constraint handling techniques that are used in our proposed method are reviewed and some background information about existing methods which deal with COPs are provided.

ε-Level constraint handling (ELCH) was first proposed in [2]. In ε-level constraint handling (ELCH), the *constraint violation value* for the solution x is defined as follows:

$$G(x) = \sum_{i=1}^{q} max\{0,\ g_i(x)\}^k + \sum_{i=1}^{p} |\ h_i(x)|^k \tag{15}$$

where k is a constant (normally set to 2). In this paper, the space of $G(x)$ vs. x is called the *constraint violation space*. Each solution x is represented by the pair (f, G) where f is the objective value at the point x and G is its constraint violation value. If f_1 and f_2 are the objective values and G_1 and G_2 are constraint violation values of the solution points x_1 and x_2, respectively, then the ε level comparison operator $\le_\varepsilon$ is defined as

$$\vec{x}_1 \le_\varepsilon \vec{x}_2 \equiv \begin{cases} f_1 \le f_2 & \text{if } G_1,\ G_2\ \le \varepsilon \text{ or } G_1 = G_2 \\ G_1 \le\ G_2 & \text{otherwise} \end{cases} \tag{16}$$

In other words, the ε-level comparison first compares two solutions by constraint violation value first. If the constraint violation value of both solutions is less than a small threshold ε, two solutions are then ranked by the objective function values only. Although there are many adaptive mechanisms to control the value of ε [33], in our paper, the value of ε is set to 0 in all experiments. Note that with this setting ($\varepsilon=0$) ELCH becomes similar to the proposed comparisons rules [64].

Gradient mutation [2,36] is a variation operator which uses gradient of the constraint functions at the solution points to guide the mutation of the candidate solutions to achieve a better constraint value. In gradient mutation, constraints array $C(x)$ and $\Delta C(x)$ are defined as

$$C(x) = (g_1(x),\ \ g_2(x), \ldots, g_m(x))^T \tag{17}$$

$$\Delta C(x) = (\Delta g_1(x),\ \ \Delta g_2(x), \ldots, \Delta g_m(x))^T \tag{18}$$

where $\Delta g_i(x) = max\{0,\ g_i(x)\}$. The value of $\Delta g_i(x)$ has been set to $max\{0,\ g_i(x)\}$ because the aim of this operator is to assist the algorithm to satisfy the constraints (i.e. $g_i(x) \le 0$). The gradient matrix of $C(x)$ is calculated [2] by the following equation:

$$\nabla C(x) = \frac{1}{\eta}(C(x+\eta e_1) - C(x), \ldots, C(x+\eta e_D) - C(x)) \tag{19}$$

where e_i's are the standard basis of the D-dimensional space. It has been shown [2] that

$$\vec{\Delta x} = \ -\nabla C(x)^{-1} \Delta C(x) \tag{20}$$

where $\vec{\Delta x}$ is an increment for x to satisfy constraints. Note that, in the cases where $\nabla C(x)$ is not invertible, numerical approaches [65] for approximating the inverse matrix can be used.

The new value for x is calculated as

$$x^{new} = x^{old} + \vec{\Delta x} \tag{21}$$

A modified version of the gradient mutation was proposed in [45] in which $\vec{\Delta x}$ was mutated before it is added to x^{old}. In fact, Eq. 21 was written as

$$x^{new} = x^{old} + m\left(\vec{\Delta x}\right) \tag{22}$$

The operator m is a mutation operator. It was shown [45] that this modified gradient (called MG) is more effective than the original one in finding quality solutions.

Many evolutionary-based methods have been proposed so far to deal with COPs. Many of these methods have been reviewed in [66] and [67]. In the following we concentrate just on a few methods which are relevant to our proposed method discussed further in the paper. Takahama and Sakai proposed a DE-based method which used ELCH and gradient mutation to handle the constraints (called εDE) [36]. By applying ELCH mechanism, εDE can handle COPs. Also, by adding gradient mutation, the method is able to solve the cases with equality constraints faster. In the newer version of εDE (called εDEag) [2], Takahama and Sakai extended εDE by using an archive that controlled the diversity of individuals which caused higher stability. εDEag also included a new scheme to control the ε level parameter. Benchmark results showed that εDEag could find quality solutions. This method received the first rank in the competition of CEC2010 in solving COPs [44].

It was also proposed [46] to use a few sub-populations of individuals and apply different mutation and crossover operators to the solutions in each sub-population. Then, the best individuals of the sub-populations are exchanged and the operators are applied again. Also, during the run, the size of each of these sub-populations (four sub-populations were used in reported experiments) is changed according to effectiveness of the operators, i.e. how good the sub-population is improving. Results indicated that this method can solve benchmark COPs effectively.

Covariance matrix adaptation evolution strategy (CMA-ES) was extended to deal with COPs [45,68]. A simple method proposed in [45] that incorporated the ELCH approach into CMA-ES to enable the algorithm to deal with constraints. In fact, ELCH was used to compare and sort the individuals in CMA-ES and allowed the algorithm to handle constraints (this variant is called ECMA-ES).

4. The proposed method, its analysis and extensions

In this section some aspects of locating disjoint feasible regions of a COP are described. Then, a PSO variant (called Epsilon Adaptive mutation linear PSO, EAPSO) [11] proposed by the authors of this paper is extended by a time-adaptive topology. By using the proposed time adaptive topology, EAPSO is not only able to locate disjoint feasible regions, but it is also able to identify the best region amongst the found regions in terms of objective value. Two local search methods, the modified gradient mutation technique (MG) and the sequential quadratic programming (SQP) (see Section 3), are also added to the proposed method to enhance its ability in finding better solutions (the new method is called EAPSO-MG). A hybrid method is proposed in which the extended EAPSO-MG is combined with ECMA-ES to find better solutions.

4.1. Locating feasible regions

According to the definition of COPs (Eq. 3) and the constraint violation function (Eq. 15), one can define a COP as follows:

$$\text{Find } x \in S \subseteq R^D \text{ such that } \begin{cases} \forall y \in \mathcal{F}\ f(x) \le f(y) & (a) \\ G(x) = 0 & (b) \end{cases} \tag{23}$$

where $G(x) = \sum_{i=1}^{q+p} \max\{0,\ g_i(x)\}^k$ (see also [69] for alternative definitions). To visualize the function $G(x)$, consider an example COP as follows:

$$f(x) = 2{*}x^2 - 1,\ g_1(x) = \ \sin(x/1.3) - .1 \le 0,\ g_2(x) = 0.02x - .1 \le 0$$

Fig. 4 shows constraint violation function ($G(x)$) for this COP together with $g_1(x)$ and $g_2(x)$.

Clearly a solution x is feasible if and only if it is in the intervals (regions) where $G(x) = 0$ (see Fig. 4). The aim of solving a COP is to find feasible x's ($G(x) = 0$) such that the objective value is minimized. However, there might be many feasible regions and the qualities of solutions in terms of the objective value in these regions are unknown. Thus, it is valuable to locate as many feasible regions as possible and then investigate which of the found regions contains higher quality solutions. By locating a region we

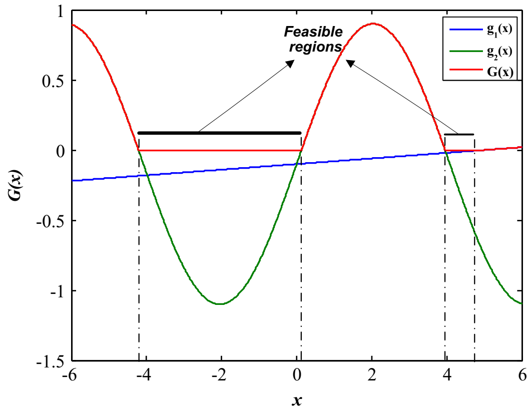

Fig. 4. A sample COP including constraints and constraint violation curves.

refer to finding at least one point in that region. Note that, locating all disjoint feasible regions in a COP corresponds with locating all niches of the function $G(x)$ (see Fig. 4). Thus, in order to locate disjoint feasible regions it is sufficient to locate niches of $G(x)$. However, because niches of $G(x)$ are flat regions with unknown sizes, it is hard to determine if two solutions are located in the same niche. Hence, if a niching method is used to locate these regions, we can only claim that the found niches are *potentially* disjoint. Throughout the paper, a feasible region which contains the optimal solution is called *optimal region*.

In our earlier work [11] we investigated locating potentially disjoint feasible regions using an evolutionary algorithm. In that paper the importance of locating potentially disjoint feasible regions and its relations with niching was discussed. Also, an extension of MLPSO with an adaptive mutation operator (called Adaptive Mutation PSO, AMPSO) was proposed that was able to locate niches very well. The velocity rule for AMPSO was written as follows:

$$\vec{v}^{\,i}_{t+1} = m(\omega\vec{v}^{\,i}_{t} + \varphi_1 r_{1t}(\vec{p}^{\,i}_{t} - \vec{x}^{\,i}_{t}) + \varphi_2 r_{2t}(\vec{g}_{t} - \vec{x}^{\,i}_{t}),\ c, \gamma^i_t) \quad (24)$$

The parameters ω, φ_1, and φ_2 are exactly the same as the ones in MLPSO. Note that the mutation operator m addresses both line search and stagnation issues if c and γ^i_t are guaranteed to be non-zero [45] (see also Section 2.4). Also, note that the vector $\vec{g}_t$ is changed to $\vec{lb}^{\,i}_t$ if a topology like ring or non-overlapping was used. The value of γ^i_t for a particle i at the time t is calculated by

$$\gamma^i_{t+1} = \begin{cases} 2*\gamma^i_t & \text{if } s^i_t > s \text{ and } \gamma^i_t < \gamma_{max} \\ 0.5*\gamma^i_t & \text{if } f_{min} < f^i_t < f_{max} \text{ and } \|\vec{v}^{\,i}_t\| < \gamma^i_t \\ 2*\gamma^i_t & \text{if } f^i_t > f_{max} \text{ and } \gamma^i_t < \gamma_{max} \text{ and } \mathrm{mod}(t,q) = 0 \\ \gamma^i_t & \text{otherwise} \end{cases} \quad (25)$$

where s^i_t (f^i_t) is the number of successive iterations that the personal best of the particle i has been (has not been) improved by at least imp_{min} percent. The value of imp_{min} was set to $1E-5$ in all experiments. At each iteration, if the personal best of the particle i was improved, s^i_t was increased by one and f^i_t was set to 0 and if it was not improved, f^i_t was increased by one and s^i_t was set to 0. If s^i_t was larger than the threshold s (equal to 10 in all experiments), the value of γ^i_t was multiplied by 2. The reason behind this multiplication was to give the algorithm the opportunity to sample further locations to improve faster. Also, if f^i_t was larger than f_{min} and smaller than f_{max}, the value of γ^i_t was reduced to enable the algorithm to conduct local search around current solutions and improve them. However, the strategy of controlling γ^i_t was reversed when the value of f^i_t was even larger than f_{max} and, consequently, γ^i_t starts to grow. The idea behind reversing the strategy was that if the current solution is not improved for a large number of successive iterations, most likely there is no better solution in the current region. Thus, it is better to jump out from the current local optima and explore the search space for other basins of attractions. According to Eq. 25, the value of γ^i_t is increased in a low rate (it is grown every q iterations) in this situation (when f^i_t is very large) to prevent the algorithm to start jumping with big steps. The values of γ_{max}, γ_{min}, f_{min}, s, f_{max}, q, c, and γ^i_0 were set to 1, $1E-10$, 10, 10, 200, 50, $\frac{1}{D^{1.5}}$, and 1, respectively, for all particles (these parameters were set by trial and error for two standard optimization functions, Sphere as a unimodal and Griewank as a multimodal function). It was shown that AMPSO is effective in niching when it uses $n-\upsilon_2$ topology. AMPSO was used to locate niches of the constraint violation function ($G(x)$) in COPs. In fact, AMPSO was combined with ELCH so that it could be applied to COPs and locate disjoint feasible regions (this method was called EAPSO in [45]). EAPSO is investigated in more details in this paper from the topology point of view. Also, it is extended by a time-adaptive topology as well as local search methods.

In this paper, for the sake of simplicity, we only consider locating potentially disjoint regions, i.e. we discuss a method that locates potentially disjoint feasible regions. The ability of the method in finding potentially disjoint feasible regions is examined through experiments. However, one can design a method (such as function stretching [70]) to increase the probability of convergence to disjoint feasible regions.

4.2. Analysis and extension of EAPSO

In this sub-section, different topologies for EAPSO are analyzed. Then, EAPSO is extended with a new time-adaptive topology which enables the algorithm to perform well in locating feasible regions as well as identifying feasible region which potentially contain high quality solutions in terms of objective value.

4.2.1. Topology of EAPSO

In order to test the ability of EAPSO with different topologies, i.e. gbest ($n-\upsilon_n$), ring, and non-overlapping topologies, in locating potentially disjoint feasible regions as well as minimizing the objective function, we used a test function, introduced in [11], called *six circles* which has one constraint (see Eq. (25)).

$$f(x) = \sum_{i=1}^{D} (x_i - 1.5)^2$$

$$\text{subject to } g(x) = \min(C_1\text{–}C_6) \le 0 \quad (26)$$

where $C_1 = \sum_{i=1}^D (x_i-1.5)^2 - 1$, $C_2 = \sum_{i=1}^D (x_i+1)^2 - 0.25$, $C_3 = \sum_{i=1}^D (x_i+3)^2 - 0.0625$, $C_4 = \sum_{i=1}^D (x_i+2)^2 + 10^{-5}$, and $C_5 = \sum_{i=1}^D (x_i-3.5)^2 + 10^{-5}$, $C_6 = \sum_{i=1}^D (2x_i)^2 + 10^{-5}$. The graph $g(x)$ vs. x has been shown in Fig. 5 for a one dimensional x.

Clearly, the function $g(x)$ has three disjoint feasible regions (centered at $x=1.5$, $x=-1$, and $x=-3$), respectively, in which $G(x)=0$ (constraint violation function). However, there are three trap regions (centered at $x=-2$, $x=0$, and $x=3.5$) where $G(x)$ reduces rapidly, but only down to 10^{-5} (note that if $G(x)=10^{-5}$ then x is not feasible). We apply the proposed EAPSO with different topologies: gbest ($n-\upsilon_n$), ring, and non-overlapping to the six circles function. For the non-overlapping topology, we test the algorithm with $n-\upsilon_6$, $n-\upsilon_4$, $n-\upsilon_3$, and $n-\upsilon_2$ (i.e. 6, 4, 3, and 2 particles in each sub-swarm). In these tests we set the maximum number of iterations (t_{max}) to $3000D/n$ and $D=10$. Also, we set

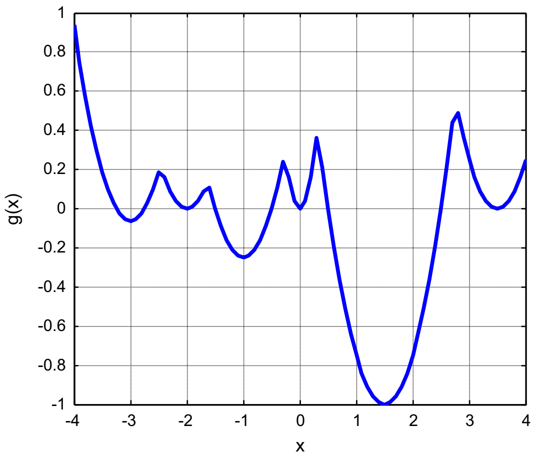

Fig. 5. One dimensional constraint space for the six circles function.

Table 1
Comparison of different topologies in EAPSO and ECMA-ES for solving the six circle function $D=10$. Note that the topology $n-v_n$ corresponds with gbest topology.

Algorithm		$D=10$						
Topology		Ring	Non-overlapping					ECMA-ES
			$n-v_n$	$n-v_6$	$n-v_4$	$n-v_3$	$n-v_2$	
Satisfaction (%)		76	58	78	95	96	100	64
Average number of feasible regions		1.17	1	1.27	1.4	1.65	2.06	1
Percentage of locating optimal region	Precision $1E-5$	26	23	28	41	53	58	37
	Precision $1E-27$	11	15	9	12	15	16	34
Percentage of pushing the solution to optimality		42	65	32	29	28	27	91

Table 2
Comparison of different topologies in EAPSO and ECMA-ES for solving the six circle function $D=30$.

Algorithm		$D=30$						
Topology		Ring	Non-overlapping					ECMA-ES
			$n-v_n$	$n-v_6$	$n-v_4$	$n-v_3$	$n-v_2$	
Satisfaction (%)		77	61	77	88	98	100	82
Average number of feasible regions		1.18	1	1.26	1.48	1.6	2.14	1
Percentage of locating optimal solution	Precision $1E-5$	27	24	31	42	50	73	39
	Precision $1E-27$	21	20	30	28	32	47	37
Percentage of pushing the solution to optimality		78	83	77	66	64	64	94

$n=12$ when $D=10$ to ensure that the swarm size is divisible by 2, 3, 4, and 6. Table 1 shows the average of the results over 100 runs.

The row "Satisfaction" shows the percentage of the runs that a feasible solution was found (e.g., PSO with ring topology has found a feasible solution in 76% of all runs). The row "Average number of feasible regions" is the number of disjoint feasible regions that was located by the particles (the number of disjoint feasible regions where the personal best of at least one of the particles is inside them at the end of the run) in the swarm in average over all runs (e.g., EAPSO with ring topology found 1.17 feasible regions in average). The row "Percentage of locating optimal region" indicates the percentage of the runs that the algorithm has found optimal region. This row contains two parts, precision $1E-5$ and $1E-27$. These two parts refer to the percentage that the algorithm has found the optimal region and its distance from the optimal solution is smaller than $1E-5$ or $1E-27$ at the end of the run. Note that the solutions that have the precision $1E-27$ also appear in the row corresponding to the "Precision $1E-5$". Comparing the results, it is clear that EAPSO with non-overlapping topology with 2 particles in each sub-swarm ($n-v_2$) has the best performance in satisfying constraints (100%), locating different feasible regions (2.06 feasible regions in average over all 3 existing regions), and finding the optimal region (58% of runs). Note that the last two measures (average of feasible solutions and percentage of locating optimal region) are interrelated as being able to find different disconnected feasible regions improves the probability of finding the one that contains optimal solution.

Let's analyze the ability of the algorithm for improving the final feasible solution (percentage of pushing the solution to optimality). Obviously, when EAPSO with the gbest ($n-v_n$) topology finds the region that contains optimal solution, in 65% ($100*15/23\%$) of the cases it could improve the solution to become closer to the optimal solution (closer than $1E-27$). The performance of different topologies in improving the final solution becomes poorer as the size of the sub-swarms becomes smaller. This is actually expected as the topologies with more particles in each sub-swarm have more resources (in this case particles) to assign to exploit better solutions around the quality existing ones. Hence, as it was expected, the gbest ($n-v_n$) topology is better than the other topologies to improve the final solution. Table 2 shows the same results when $D=30$ and $n=12$.

Also, the results in 30 dimensional space confirm the results of 10 dimensional space. In fact, a better performance in locating feasible regions appears when the size of the sub-swarms is small while better performance in improving the final solutions appears when the size of the sub-swarms is large.

4.2.2. A time-adaptive topology for EAPSO

As it was discussed in Section 4.2.1, EAPSO is able to locate potentially disjoint feasible regions with $n-v_l$ where l is small. However, it is important to find which of the located regions contain higher quality solutions (in terms of objective function). Also, according to the results in Section 4.2.1 and based on this fact that the algorithm should exploit better solutions at the latter stages of the optimization process, it is proposed to change the topology over time from several small sub-swarms ($n-v_l$ with small l) to the $n-v_n$ topology. Topology based on several small sub-swarms ($n-v_2$ for example) explores the search space very well, while $n-v_n$ is more effective to exploit better solutions close to the existing quality ones. We propose to start the optimization process with $n-v_l$ topology with a small $l=l_t=l_{min}$ and then the value of l_t is increased to l_{max} during the run. This increment is done linearly (according to t) until a proportion of the maximum allowed iterations (t_{max}) is reached, i.e. $t=k*t_{max}$. In fact the value of l ($n-v_l$) is updated by the iteration number (t) as follows:

$$l_t = \begin{cases} \left\lfloor \left(\frac{l_{max}-l_{min}}{t_{max}-1}\right)(t-1)+l_{min} \right\rfloor & \text{if } t<k*t_{max} \\ n & \text{otherwise} \end{cases} \tag{27}$$

where t is the iteration number, n is the swarm size, and k is a real number in the interval [0, 1]. By using this formulation, the value of l_t (the number of particles in each sub-swarm at each iteration) is l_{min} at the beginning of the optimization process and it grows to l_{max} at the iteration $t=k*t_{max}$. As n might not be divisible by l_t, we consider the size of the first sub-swarm at iteration t equal to $l_t+(n \bmod l_t)$ where *mod* is the modulo operator. By using this value for the size of the first sub-swarm, the remaining number of particles (i.e. $n-(l_t+(n \bmod l_t))$) is always divisible by l_t. Note that, in this case, l_t is always larger than 1, which is desirable. Note also that if $l_t > n/2$ then the size for the first sub-swarm becomes equal to n (because $n \bmod l_t$ for $l_t > n/2$ is equal to $n-l_t$) which results in the $n-v_n$ topology. If n is divisible by l_t, $n \bmod l_t$ becomes zero and the size of the first sub-swarm becomes l_t. According to the results reported in the Section 4.2.1, the value $l_{min}=2$ is considered. Also, l_{max} is set to $n/2$ because any value greater than $n/2$ results in $l_t=n$.

To set the value of the parameter k, we applied the PSO with the proposed time adaptive topology to the six circle function with different number of dimensions and different values for k. We applied the algorithm 50 times to the test function. The number of FE was set to 500D in all tests. Also, in all tests, the values of l_{min} and l_{max} were set to 2 and $n/2$ respectively. Fig. 6 shows the results.

Fig. 6 shows the average of results together with the standard deviation for different number of dimensions. From Fig. 6, the value $k=0.75$ seems have good performance in terms of the average of the quality found solutions. Thus, we propose to set $k=0.75$, $l_{min}=2$, and $l_{max}=n/2$, according to the above parameter setting. In this case, the algorithm uses non-overlapping topology at the first 75% of iterations and the gbest topology at the remaining 25% of all allowed iterations.

Fig. 7 shows how the value of l is changed during the run when $n=20$ and $t_{max}=100$.

This time-adaptive topology is used in all further tests for EAPSO. The time adaptive approach enables several small sub-swarms searching for local optima at the beginning of the optimization process. During the run, some particles in some sub-swarms might join other sub-swarms as soon as the value of l_t is changed. A newly joined particle to a sub-swarm might be the new lead or might be a new follower. Either way, the resources allocated to each region is increased during the time and, at the same time, better particles (particles which their personal best has higher quality) will have more followers. Thus, more particles are assigned to the regions where a higher quality solution has been located in.

To clarify how topology is changed during the optimization process, consider a swarm with six particles in a search space.

In Fig. 8 the lead particles have been shown by white dotes and follower particles have been shown by black dots. Consider that the topology is $n-v_2$ (Fig. 8(a)) at iteration t, i.e. there are three sub-swarms: $S_t^1=\{1, 2\}$, $S_t^2=\{3, 4\}$, and $S_t^3=\{5, 6\}$. Now, assume that the topology is going to change to $n-v_3$ (Fig. 8(b)) at iteration $t+1$. In this case, there are two sub-swarms: $S_{t+1}^1=\{1, 2, 3\}$, and $S_{t+1}^2=\{4, 5, 6\}$. As the particle 3 is better than particles 1 and 2 (it is feasible and the value of f for the particle 3 is lower than both particles 1 and 2), particle 3 will be the leader of the sub-swarm S_{t+1}^1. This actually results in attracting particles 1 and 2 towards the region A in next iterations. Also, particle 5 will be the leader for the sub-swarm 2, which results in attracting particle 4 towards region C.

The proposed time-adaptive topology is used in EAPSO (and all of its derivatives) in all further tests and comparisons.

4.2.3. Gradient mutation and local search

The gradient mutation is used to direct the candidate solution x towards feasible area $\mathcal{F}$. It is proposed to use modified gradient mutation, MG, in combination with EAPSO (this is called EAPSO-

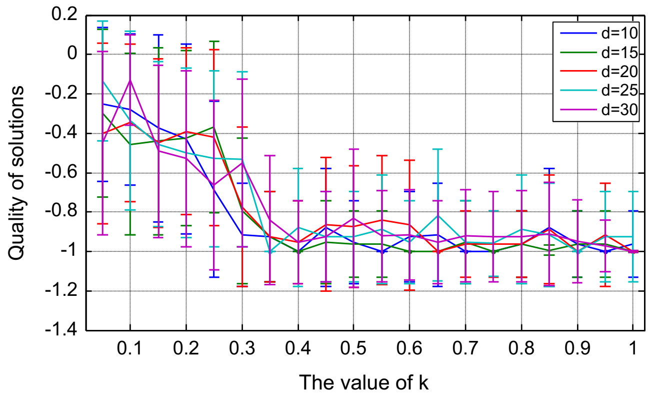

Fig. 6. Setting the parameter k.

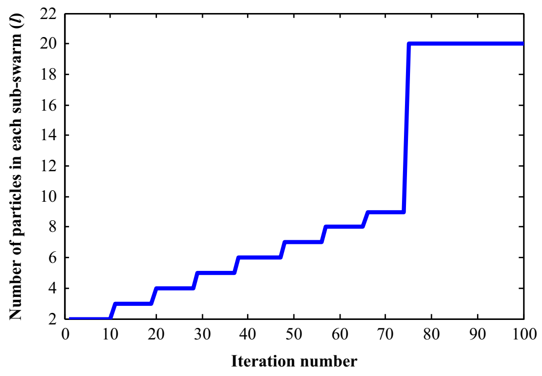

Fig. 7. Changes of the number of particles in each sub-swarm in each iteration.

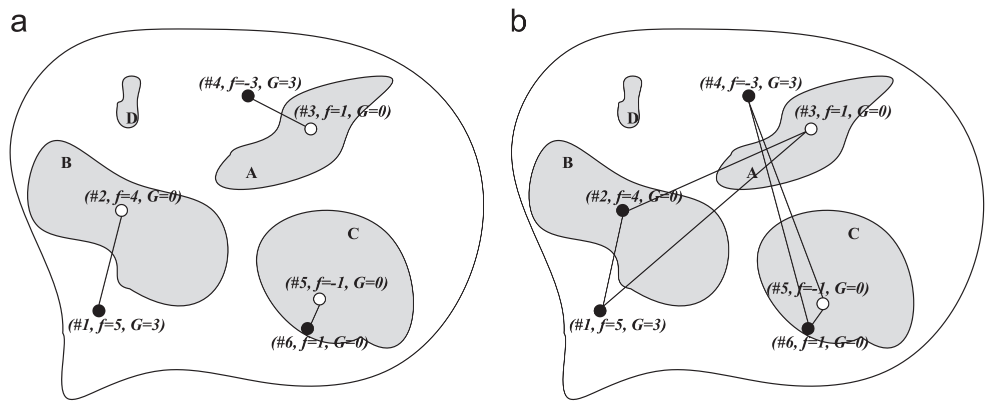

Fig. 8. Both figures show six particles in a search space with some feasible regions (gray areas). Particles have been shown by dots in the search space; black dots are followers and white dots are the leaders. Particles have been connected through (a) $n-v_2$ topology (iteration t), and (b) $n-v_3$ topology (iteration $t+1$). The triples close to each particle shows (particle number, objective value, and constraint violation value).

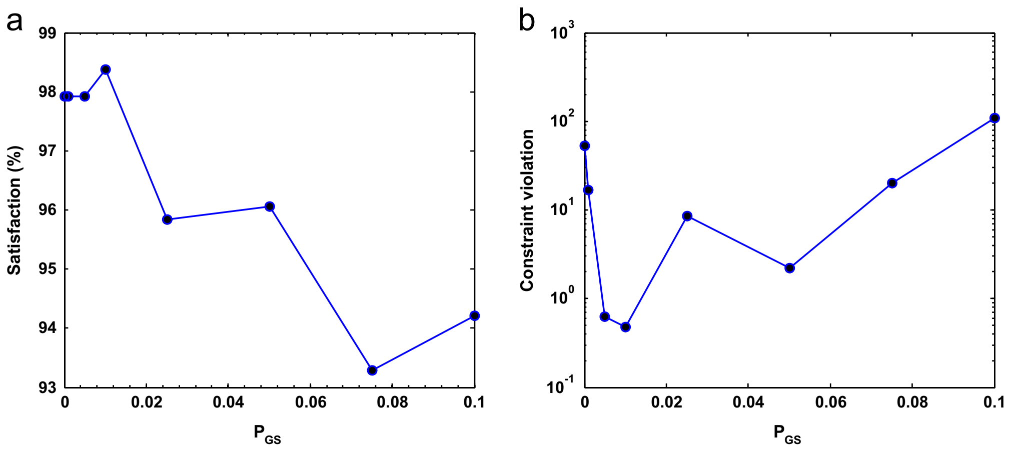

Fig. 9. Performance of EAPSO-MG with different values for P_G (a) the average percentage of constraint satisfaction, and (b) the average of constraint violation. The averages are over all 18 functions with 25 runs each.

MG) to improve its ability for satisfying constraints. Indeed, Eq. 24 is re-written for EAPSO-MG as follows:

$$\vec{v}^{\,i}_{t+1} = \begin{cases} m(\Delta\vec{x}_t,\ c, \gamma^i_t) & rand < P_G \\ \text{Eq. 24} & \text{otherwise} \end{cases} \tag{28}$$

where r *and* is a uniform random number generated from the interval [0, 1], P_G is the probability that the velocity vector follows the gradient direction and $1-P_G$ is the probability that it follow the standard formulation (Eq. (24)).

To set the parameter P_G, EAPSO-MG was applied to 5 randomly selected functions from the CEC2010 [44] benchmark when P_G was changed from 0 to 1. The algorithm was terminated once it could locate the first feasible solution. Also, in this test, D was set to 10 and n was set to 20. Fig. 9 shows the results (average percentage of satisfying constraints and average of constraint violation value).

Note that, because the performance of the algorithm for $P_G > 0.1$ impaired dramatically, these values for P_G have not been excluded from the figure. The reason for this impairment is that large values for the gradient search causes particles to move to local optima at the early stage of the optimization process. From the figure, it seems that $P_G=0.01$ has the best performance both in terms of constraint satisfaction and constraint violation. Hence, this value is used hereafter for P_G throughout the paper.

In order to improve the quality of solutions (in terms of objective value) found by EAPSO-MG, the local search (SQP) that was introduced in [3] is also used in our approach. SQP is applied to the personal best of a randomly selected particle within the swarm with the probability P_L and for maximum number of iterations equal to t_L (set to $100D$ in all experiments, where D is the number of dimensions). Because the local search mostly concentrates on minimizing the objective value, we propose to do not apply the SQP local search on infeasible solutions. We also set the value of P_L to 0.01 in all experiments. This value was set experimentally the same as setting procedure for P_G. Any method which uses this local search in this paper is postfixed by L (e.g., EAPSO-MG which uses this local search is called EAPSO-MGL).

The following algorithm shows the procedure for EAPSO-MGL.

```
Algorithm EAPSO-MGL
Inputs: n, D, TC (termination condition)
Outputs: global best particle
Set $\gamma_{max}, \gamma_{min}, f_{min}, s, f_{max}, q, c, P_L, P_G$, and $\gamma^i_0$ to 1, 1E−10, 10, 10,
   200, 50, $\frac{1}{D^{1.5}}$, 0, 0.01, and 1
t=0, TC=false
Initialize $\vec{x}^{\,i}_t, \vec{v}^{\,i}_t, \vec{lb}^{\,i}_t$, and $\vec{p}^{\,i}_t$ for all particles
Initialize topology for all particles to $n-v_2$
While TC is false
  For $i=1:n$
    Calculate $\vec{v}^{\,i}_{t+1}$ by Eq. 28
    Use standard position update rule
  End for
  Update topology of the swarm according to Eq. 27 (k=0.75,
  $l_{min}=2$, and $l_{max}=n/2$)
  Update $\vec{p}^{\,i}_t, \vec{lb}^{\,i}_t$ according to the quality of particles (Eq. 16)
  Update $\gamma^i_{t+1}$ according to Eq. 25
  If at least one feasible solution was found
    Update $P_L=0.01$
  End if
  If rand < $P_L$
    Randomly select one of the particles and apply local search
  (SQP) on its personal best
  End if
  $t=t+1$
  Update TC
  End while
  Return the particle which has the best personal best among
 the swarm
  End of the algorithm
```

4.3. Termination conditions

Four termination conditions (TC) are considered in this paper in different experiments for different algorithms. In each experiment in the next sections, a logical disjunction of some of these conditions is used for terminating the algorithm.

TC1(inp): this is true when the number of iterations is equal to *inp*.
TC2: this is true when the first feasible location is met.
TC3(inp): this is true when the global best vector was not improved for the last inp successive iterations.
TC4(inp): this is true when the method is run for *inp* number of iteration after finding the first feasible solution.

TC1 terminates the algorithm when a predefined number of iterations (*max_itr*) is achieved. By using TC2, the algorithm is terminated when the first feasible solution is found. By using TC3, the algorithm is stopped when it could not find any better solution in a predefined number of evaluations (t_f). This condition is beneficial to recognize if the particles have got stuck in local optima. TC4 is useful for testing the ability of the methods in improving objective values after finding the first feasible solutions.

4.4. Comparisons of EAPSO, EAPSO-MG, and ECMA-ES

In this sub-section, two main comparisons are conducted which test the ability of the proposed methods (EAPSO and EAPSO-MG) in satisfying constraints and optimizing objective value.

4.4.1. Satisfying constraints

Proposed methods (EAPSO, EAPSO-MG) are compared with ECMA-ES in terms of their abilities in satisfying constraints through two tests:

1) EAPSO, EAPSO-MG, and ECMA-ES are compared in terms of percentage of satisfying constraints.
2) EAPSO, EAPSO-MG, and ECMA-ES are compared in terms of the number of needed function evaluations (FEs) for satisfying constraints.

In all tests, we use 10 dimensional standard test functions in CEC2010 [44]. In this test, *max_itr* was set to 20,000D/n, termination condition was set to TC2 or TC1(*max_itr*) (either of them was satisfied, the method is terminated), and the population size for PSO-based methods was set to 2D and for ECMA-ES method was set to the default value proposed in [7]. All reported results are the averages over 25 runs of each method for each function.

Table 3 shows the results of applying different methods to all (18) COPs in a standard benchmark problem set in CEC2010 [44].

Table 3
Comparisons of ECMA-ES, EAPSO, and EAPSO-MG (P_G=0.01).

Row	Method	Satisfaction (%)				Constraint violation
		*f*1–*f*10, *f*13–*f*18	*f*11	*f*12	Average	
1	ECMA-ES	100	76	**88**	98	0.667269
2	EAPSO	100	80	**88**	98.222	53.29491
3	**EAPSO-MG**	**100**	**88**	**88**	**98.6666**	**0.483218**

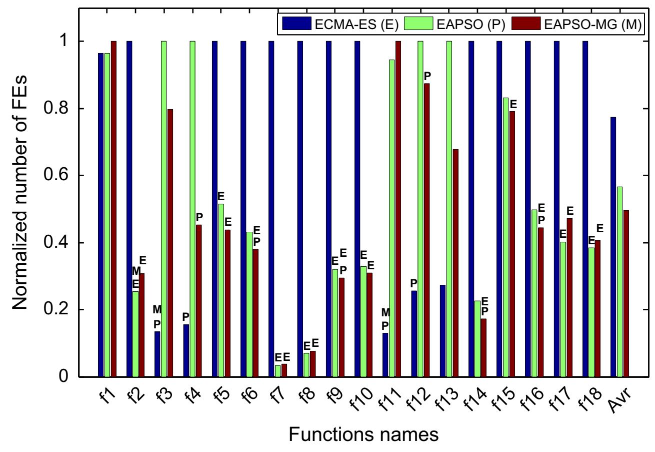

Fig. 10. The average normalized number of function evaluations to find first feasible solution.

EAPSO-MG (recall that P_G=0.01) has better performance in comparison to ECMA-ES in satisfying the constraints in average as it has satisfied constraints in larger percentage of runs (98.6 comparing to 97.9) and has smaller value for the constraint violation (0.48 comparing to 0.66). However, based on Wilcoxon test with $p<0.05$, these results were not statistically significant (the p value for the test for comparing ECMA-ES and EAPSO was 0.29, for comparing ECMA-ES and EAPSO-MG was 0.37, and for comparing EAPSO-MG and EAPSO was 0.22).

Because the percentages of satisfactions are very close to each other and the differences are not statistically significant, we also compare the number of iterations needed to find the first feasible solution (Fig. 10). The number of FEs has been normalized to maximum among the methods in each case. Also, the Wilcoxon test (with $p<0.05$) was used to test if the differences between the results are significant. Each method that has significantly better performance (in terms of the number of FE) than other methods has been marked by one the three letters E (significantly better performance than ECMA-ES), P (significantly better performance than EAPSO), or M (significantly better performance than EAPSO-MG).

The results reported in Fig. 10 indicate that

- ECMA-ES is significantly faster than EAPSO in *f*3, *f*4, *f*11, and *f*12 (4 cases) and significantly faster than EAPSO-MG in *f*3 and *f*11 (2 cases);
- EAPSO is significantly faster than ECMA-ES in *f*2, *f*5, *f*6, *f*7, *f*8, *f*9, *f*10, *f*17, and *f*18 (9 cases) and significantly faster than EAPSO-MG in *f*2 (1 case);
- EAPSO-MG is significantly faster than EAPSO in *f*4, *f*6, *f*9, *f*12, *f*14, and *f*16 (6 cases) and significantly faster than ECMA-ES in *f*2, *f*5, *f*6, *f*7, *f*8, *f*9, *f*14, *f*15, *f*16, *f*17, and *f*18 (11 case).

Thus, it is obvious that EAPSO-MG performs better than ECMA-ES and EAPSO in terms of percentage of satisfying constraints (Table 3, not statistically significant though) and the number of FEs needed to find the first feasible solution (Fig. 10, statistically significant in most cases).

4.4.2. Optimizing objective function

In this sub-section EAPSO-MG and ECMA-ES are compared in terms of their ability in optimizing objective function. Each method is applied to all COPs (18 functions) from CEC2010 benchmark 25 times. The value of *max_itr* was set to 20,000D/n, number of dimensions was set to 10 (D=10), population size (n) for EAPSO-MG was set to 2D, and population size for ECMA-ES was set to $4+3*\ln(D)$ (the same as [7]). The termination condition was

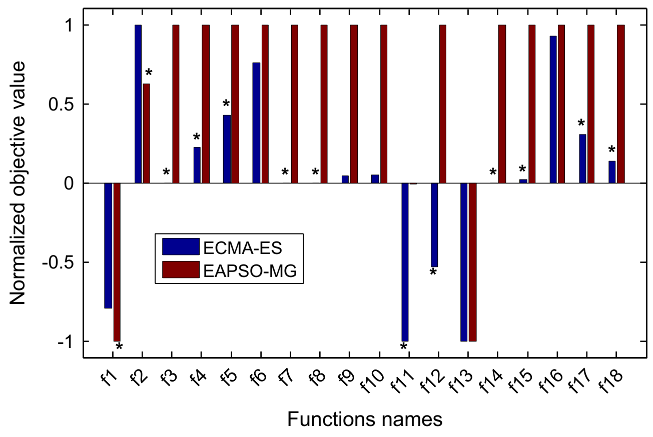

Fig. 11. The average normalized objective value in 500D/n iterations found by ECMA-ES and EAPSO-MG, the smaller the better.

TC4($500D/n$) or TC1(max_itr). In fact, the methods were run $500D/n$ iterations after the first feasible solution was found or the maximum number of iterations is reached. Fig. 11 shows the average of normalized objective value over 25 runs. As we consider the minimization problems only, the lesser the value of normalized objective function is, the better the method is.

From the figure, it is obvious that ECMA-ES performs significantly better (based on the Wilcoxon test with $p < 0.05$, asterisked in the figure) than EAPSO-MG in minimizing the objective value in 12 cases ($f3$, $f4$, $f5$, $f7$, $f8$, $f11$, $f12$, $f14$, $f15$, $f17$, and $f18$) over all 18 cases.

4.5. The hybrid method

It was shown (Section 4.4) that EAPSO-MG is more effective than ECMA-ES for satisfying constraints while ECMA-ES is more effective in optimizing objective function. Thus we propose a hybrid method which contains two phases: starting with EAPSO-MGL to satisfy constraints and then running ECMA-ES to improve the final solutions (this hybrid method is called EAPSO-CMA). This hybridization is also beneficial as EAPSO-MG has a better ability to locate different feasible regions (comparing to ECMA-ES), which improves the overall capability of the method to locate the optimal solution. Termination condition for EAPSO-MGL is set to $TC_1(t_{P1})$ or $TC_3(t_f)$, i.e. t_{P1} iterations for the first phase and t_f is the number of successive iterations that the global best was not improved, and then ECMA-ES is run with the termination condition $TC_1(t_{P2})$ where $t_{P2}=max\_itr\text{-}t_{P1}$. Because ECMA-ES is used to improve the final solutions found by EAPSO-MGL, the number of iterations assigned to ECMA-ES (t_{p2}) is set to a smaller value comparing to that of EAPSO-MGL (t_{p1}).

To set the value of t_{P1}, we run some experiments on four randomly selected benchmark functions from CEC2010 ($f1$, $f5$, $f7$, and $f16$). EAPSO-CMA was applied to these functions for $20{,}000D/n$ iterations with $t_{P1} = r \times max\_itr$, where $r \in [0, 1]$. We changed the value of r from 0 to 1, with the step size 0.05, and recorded the average normalized objective value (normalized to $[-1, 1]$).

Fig. 12 Shows that for r values from the interval 0.4–1, the performance of EAPSO-CMA is better than other values for r. Thus, in our experiments, the value of r was set to 3/4.

Also, in our experiments, we noticed that if termination of EAPSO-MGL was caused by $TC_3(t_f)$, ECMA-ES also could not improve the final solutions. The reason is that when EAPSO-MGL could not find better solutions after long number of iterations, it was highly probable that the best found solution was already at a local optimum. Thus, it is not expected that ECMA-ES in the next phase can find any better solutions around the current best found solution. Hence, it is better to rerun the EAPSO-MGL to search for other basins of attractions. Our experiments showed that $t_f=1000$ is a good choice for decision on rerunning EAPSO-MGL.

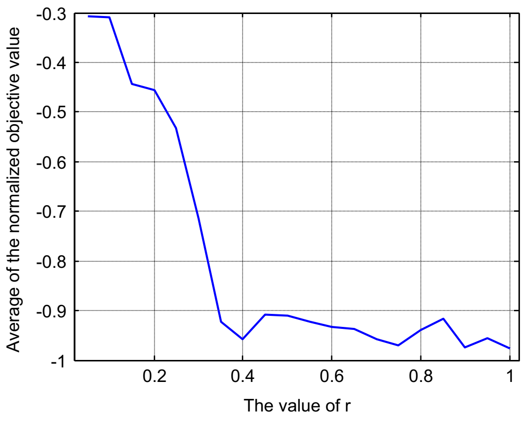

Fig. 12. The result of setting the value of r (the smaller the better).

ECMA-ES is initialized by $4+3*\ln(D)$ points (the standard formulation for the population size of CMA-ES presented in [43]) generated by a multi-variant normal distribution with the variance $\rho=1E-3$ and the mean equal to the best found solution by EAPSO-MGL. Note also that the value of ρ was set to a small number to ensure that ECMA-ES is concentrated on a small area around the solution found in the previous phase. This hybrid method is called EAPSO-CMA.

```
Algorithm EAPSO-CMA
  Inputs: $\rho$, max_itr
  Outputs: best solution
    $t_{p1}=3max\_itr/4$
    Old_best_found=run Algorithm EAPSO-MGL with D (number of dimensions of the problem), $n=2D$, and TC=TC1 ($t_{P1}$) or TC3($t_f$)
  If Algorithm EAPSO-MGL was stopped because of TC3($t_f$)
      New_best_found=Algorithm EAPSO-MGL with $D$ (number of dimensions of the problem), $n=2D$, and TC=TC1 ($3max\_itr/4-t_{P1}$)
      Best_found= best solution between Old_best_found and New_best_found
Else
      Best_found=Old_best_found
End if
  Generate $4+3*\ln(D)$ points by $N$(Best_found, $\rho$) and initialize ECMA-ES with those points
  Run ECMA-ES for $t_{P2}$=max_itr-$t_{P1}$ iterations
  Return the best found solution by ECMA-ES
End of the algorithm
```

Let us summarize the parameters for the proposed methods:

EAPSO-MGL: $max\_itr=20{,}000D/n$ (set for all methods), $n=2D$ (trials), $\gamma_{max}=1$ (trials and errors on two functions, see Section 2.4), $\gamma_{min}=1E\text{-}10$ (trials and errors on two functions, see Section 2.4), $f_{min}=10$ (trials and errors on two functions, see Section 2.4), $s=10$ (trials and errors on two functions, see Section 2.4), $f_{max}=200$ (trials and errors on two functions, see Section 2.4), $q=50$ (trials and errors on two functions, see Section 2.4), $c=1/D^{1.5}$ (trials and errors on two functions, see Section 2.4), $\gamma_0^i=1$ (trials and errors on two functions, see Section 2.4), $P_G=0.01$ (set experimentally, see Section 4.2.3), $P_L=0.01$ (set experimentally, see Section 4.2.3), $k=0.75$ (set experimentally, see Section 4.2.2), $l_{min}=2$ (minimum allowed swarm size, see Section 4.2.2), $l_{max}=n/2$ (set experimentally, see Section 4.2.2).
EAPSO-CMA: the values of $t_{P1}=3max\_itr/4$ (experimentally set, see Fig. 12), $t_{P2}=max\_itr-t_{P1}$ (remaining number of iterations), and $\rho=1E-3$ (trial and errors).
ECMA-ES: $\rho=1$ and number of initial points (initial population size) $=4+3*\ln(D)$, taken from the standard CMA-ES.

Note that the number of parameters in other methods (such as εDEag) is almost the same as for EAPSO-CMA. However, most of the parameters in those methods were not discussed in their original papers and they were given as constants in those papers. As an example, in the algorithm εDEag [2] (see appendix 1 for the steps of this algorithm), the crossover rate was set to 0.95, the number of DE operations was set to 2, probability of scaling factor was set to 0.05, probability of selection from archive was set to 0.05, etc. (see step 5 of the algorithm in appendix 1). However, these constants are in fact parameters of the method that may

Table 4
EAPSO-CMA, EAPSO-MGL, and ECMA-ES were applied to the COP instances from CEC2010 benchmark set. The font for the best results in each case has been bolded.

Function no.	EAPSO-CMA (*C*)	EAPSO-MGL (*M*)	ECMA-ES (*E*)
1	**−6.365E−01**^{ME}	−3.469E−01	−5.284E−01
2	**−2.276E+00**^{ME}	−2.218E+00	−2.182E+00
3	**0**^{ME}	7.892E−12	2.133E−18^{M}
4	**−1.000E−05**^{ME}	−0.818E−06	−0.929E−06^{M}
5	**−4.70E+02**^{ME}	−2.557E+02	−3.39E+02^{M}
6	**−5.73E+02**^{ME}	−4.221E+02	−4.34E+02
7	**0**^{ME}	1.428E−09	4.29E−19^{M}
8	**0**	**0**	**0**
9	**0**^{E}	**0**^{E}	3.99E−13
10	3.541E+01	4.782E+01	**4.353E−01**^{CM}
11	**−1.523E−03**^{M}	−1.224E−04	**−1.523E−03**^{M}
12	**−1.251E+02**^{ME}	−1.057E+02	−1.199E+02
13	**−5.959E+01**^{ME}	−5.384E+01	−5.722E+01
14	**2.011E+01**^{ME}	3.563E+01	4.142E+01
15	**1.490E+02**^{ME}	2.921E+03	1.771E+02^{M}
16	2.541E−02	**2.541E−03**^{CE}	3.177E−01
17	**0**^{M}	1.745E−19	**0**^{M}
18	**0**	**0**	**0**

impact its performance. Thus, the main advantage of the proposed method can be measured by the reported results.

5. Experiments and comparisons

In this section, the results of the proposed methods together with some state-of-the-art methods for solving COPs are compared. All methods were applied to standard benchmark COPs (18 standard test functions) of CEC2010.

Two tests are conducted in this section:

Test 1: the results of EAPSO-CMA are compared with the one of EAPSO-MGL and ECMA-ES. This comparison is done to show whether the hybridization has improved the overall performance of the method.

Test 2: results of the EAPSO-CMA are compared with EPSO-CMA [45], εDEag [2], and SMAOGA [46]. The aim of this comparison is to show that the results of the proposed method are competitive with the ones from other state-of-the-art methods.

The results for EPSO-CMA, εDEag, and SMAOGA were taken directly from the source papers. Note that the maximum number of FE was set to 20,000*D* for all experiments and all methods.

5.1. Test 1

In this test, the methods EAPSO-CMA, EAPSO-MGL, and ECMA-ES are applied to all 18 COP benchmarks from CEC2010. The number of dimensions was set to 10 in all tests. Table 4 shows the results. Each method was applied to each instance 25 times and the table only shows the average of the results. The Wilcoxon test (with $p < 0.05$) was used to determine the significance of the difference between the results of the algorithms. The results have been superscripted by C, M, or E. The result of each method that is significantly better than other methods has been superscripted by C (significantly better than EAPSO-CMA), M (significantly better than EAPSO-MGL), and E (significantly better than ECMA-ES).

It is clear in Table 4 that EAPSO-CMA has significantly better performance in the tested benchmark set in 11 cases than the both other methods. This means that the proposed hybridization was successful and the new hybrid method performs better than the both primary methods in solving the listed test cases.

5.2. Test 2

In this section, the results of comparison among EAPSO-CMA with three other state-of-the-art methods (EPSO-CMA, εDEag, and SMAOGA) are presented, where all methods are applied to 18 standard COPs from CEC2010 (see Table 5).

EAPSO-CMA, EPSO-CMA, and SMAOGA found feasible solutions in all runs for all functions while this was not the case for εDEag (the percentage of the number of runs which satisfied the constraints was 12% for 30 dimensional *f*12).

Comparison between EPSO-CMA and EAPSO-CMA: for 10 dimensional cases, EAPSO-CMA algorithm has the better average of objective value comparing to EPSO-CMA on 8 cases (*f*1, *f*5, *f*6, *f*10, *f*12, *f*13, *f*14, *f*15) and their results are the same for the remaining cases. Also, for 30 dimensional cases, EAPSO-CMA algorithm has the better average of objective value in comparison to EPSO-CMA on 8 cases (*f*1, *f*3, *f*5, *f*6, *f*9, *f*10, *f*13, *f*15) and their results are the same for the remaining cases. In terms of the rank, EAPSO-CMA algorithm has a better rank comparing to EPSO-CMA in both 10 and 30 dimensional test functions.

Comparison between εDEag and EAPSO-CMA: for the 10 dimensional cases, EAPSO-CMA has the better average of objective value comparing to εDEag in 6 cases (*f*2, *f*4, *f*8, *f*16, *f*17, *f*18) while εDEag has a better performance in 8 cases (*f*1, *f*5, *f*6, *f*10, *f*12, *f*13, *f*14, *f*15). However, in the 30 dimensional cases, EAPSO-CMA has the better average of objective value in 13 cases (*f*2, *f*3, *f*4, *f*7, *f*8, *f*9, *f*10, *f*11, *f*12, *f*14, *f*15, *f*17, *f*18) while εDEag has a better performance in 4 cases (*f*1, *f*5, *f*6, *f*13). In terms of the rank, EAPSO-CMA is better than εDEag in both 10 and 30 dimensional cases.

Comparison between SMAOGA and EAPSO-CMA: for the 10 dimensional cases, EAPSO-CMA has the better average of objective value comparing to SMAOGA in 12 cases (*f*2, *f*3, *f*4, *f*7, *f*8, *f*9, *f*10, *f*11, *f*12, *f*14, *f*15, *f*17, *f*18) while SMAOGA has a better performance in 4 cases (*f*1, *f*6, *f*13, *f*16). However, in the 30 dimensional cases, EAPSO-CMA has the better average of objective value in 10 cases (*f*2, *f*4, *f*9, *f*10, *f*11, *f*14, *f*15, *f*16, *f*17, *f*18) while εDEag has a better performance in 6 cases (*f*1, *f*3, *f*5, *f*6, *f*12, *f*13). In terms of the rank, EAPSO-CMA is better than SMAOGA in both 10 and 30 dimensional cases.

Note that, the proposed method does not perform well in some cases (e.g., *f*1) consistently. Analysis of this poor performance of the method is left as a potential future study.

6. Conclusions and future works

In this paper a particle swarm optimization variant, called Adaptive Mutation PSO (AMPSO), was investigated. This variant was used to locate potentially disjoint feasible regions in COPs. AMPSO was combined with a constraint handling method called epsilon level constraint handling (ELCH) and the new method was called EAPSO. Topology of EAPSO was studied in details and a time adaptive topology was proposed. It was found that a non-overlapping topology with small number of particles in each sub-swarm is beneficial in locating disjoint feasible regions. Also, topologies which contain more particles in each sub-swarm are beneficial in improving solutions in terms of the objective function. The proposed time-adaptive topology enables EAPSO to locate disjoint feasible regions at the beginning of the optimization process while concentrate on the best region to improve the solutions in terms of objective value at the latter stage of the optimization process. EAPSO was also combined with local search methods to improve its ability in locating feasible regions as well

Table 5
Results of applying EAPSO-CMA, EPSO-CMA, εDEag, and SMAOGA on all problems from CEC2010. The rank of each algorithm in each row has been indicated in {.}.

D	Function no.	EAPSO-CMA	EPSO-CMA	εDEag	SMAOGA
10	1	−6.365E−01 {3}	−3.725E−01 {4}	**−7.470E−01 {1}**	**−7.470E−01 {1}**
	2	**−2.278E+00 {1}**	**−2.278E+00 {1}**	−2.259E+00 {4}	−2.272E+00 {2}
	3	**0 {1}**	**0 {1}**	**0 {1}**	1.190E−10 {4}
	4	**−1.000E−05 {1}**	**−1.000E−05 {1}**	−9.918E−06 {4}	−9.930E−06 {3}
	5	−4.70E+02 {2}	−4.405E+02 {3}	**−4.836E+02 {1}**	−4.017E+02 {4}
	6	−5.73E+02 {3}	−5.199E+02 {4}	**−5.787E+02 {1}**	−5.777E+02 {2}
	7	**0 {1}**	**0 {1}**	**0 {1}**	**0 {1}**
	8	**0 {1}**	**0 {1}**	6.728E+00 {4}	**0 {1}**
	9	**0 {1}**	**0 {1}**	**0 {1}**	2.731E+03 {4}
	10	3.541E+01 {2}	3.743E+01 {3}	**0 {1}**	1.732E+02 {4}
	11	**−1.523E−03 {1}**	**−1.523E−03 {1}**	**−1.523E−03 {1}**	−5.260E−04 {4}
	12	−1.251E+02 {2}	−9.934E+01 {3}	**−3.367E+02 {1}**	−5.588E+01 {4}
	13	−5.959E+01 {3}	−5.369E+01 {4}	**−6.843E+01 {1}**	−6.838E+01 {2}
	14	2.011E+01 {2}	1.239E+02 {3}	**0 {1}**	3.873E+02 {4}
	15	1.490E+02 {2}	1.305E+03 {4}	**1.799E−01 {1}**	8.575E+02 {3}
	16	2.541E−02 {2}	2.541E−02 {2}	3.702E−01 {4}	**1.403E−03 {1}**
	17	**0 {1}**	**0 {1}**	1.250E−01 {4}	1.271E−02 {3}
	18	**0 {1}**	**0 {1}**	9.679E−19 {3}	1.053E−02 {4}
Average of ranks		**1.667**	2.167	1.944	2.833
30	1	−5.159E−01 {3}	−3.144E−01 {4}	**−8.209E−01 {1}**	−8.115E−01 {2}
	2	**−2.272E+00 {1}**	**−2.272E+00 {1}**	−2.151E+00 {4}	−2.252E+00 {2}
	3	7.164E+00 {2}	1.434E+01 {3}	2.884E+01 {4}	**2.255E−07 {1}**
	4	**1.024E−06 {1}**	**1.024E−06 {1}**	8.163E−03 {4}	1.471E−03 {3}
	5	−9.843E+01 {3}	−8.815E+01 {4}	−4.495E+02 {2}	**−4.716E+02 {1}**
	6	6.511E+01 {3}	2.089E+02 {4}	**−5.279E+02 {1}**	−5.208E+02 {2}
	7	**0 {1}**	**0 {1}**	2.604E−15 {4}	**0 {1}**
	8	**0 {1}**	**0 {1}**	7.831E−14 {4}	**0 {1}**
	9	**2.198E+00 {1}**	3.718E+00 {2}	1.072E+01 {3}	1.526E+04 {4}
	10	**2.672E+01 {1}**	2.975E+01 {2}	3.326E+01 {3}	6.032E+03 {4}
	11	**−3.923E−04 {1}**	**−3.923E−04 {1}**	−2.864E−04 {3}	−1.260E−04 {4}
	12	−3.461E+00 {2}	−3.461E+00 {2}	3.562E+02 {4}	**−3.493E+00 {1}**
	13	−5.849E+01 {3}	−5.473E+01 {4}	**−6.535E+01 {1}**	−6.310E+01 {2}
	14	**0 {1}**	**0 {1}**	3.089E−13 {3}	1.763E+03 {4}
	15	**1.449E+01 {1}**	1.947E+01 {2}	2.160E+01 {3}	1.750E+04 {4}
	16	**0 {1}**	**0 {1}**	**0 {1}**	3.737E−03 {4}
	17	**0 {1}**	**0 {1}**	6.326E+00 {3}	1.344E−02 {4}
	18	**0 {1}**	**0 {1}**	8.755E+01 {4}	1.052E+01 {3}
Average of ranks		**1.555**	2	2.889	2.661

as improving final solutions. Experiments showed that ECMA-ES (a CMA-ES based method for COPs) performs better than EAPSO-MG in minimizing the objective function (significantly better in 12 functions out of 18). However, EAPSO-MG was more effective (significantly faster in terms of needed function evaluations) than ECMA-ES in satisfying constraints. Thus, a hybrid method called EAPSO-CMA was proposed which used EAPSO to locate feasible regions, then concentrate on only one region, and at the end ECMA-ES is used to find better solutions in that region. EAPSO-CMA was compared with some other state-of-the-art methods in dealing with COPs. All methods were applied to standard test problems from CEC2010 benchmark. Results show that EAPSO-CMA is effective in both satisfying constraints and minimizing the objective function. As a future direction, it would be useful if the topic of locating disjoint feasible regions in a COP is investigated in more details. One example would be to design methods to guarantee locating disjoint feasible areas in a COP. Also, in some cases, the performance of the proposed method is poor. An in-depth analysis of the performance of the proposed method on these test cases can be also valuable and it is left as a future work.

Acknowledgments

This work was partially funded by the ARC Discovery Grant DP130104395 and by grant N N519 5788038 from the Polish Ministry of Science and Higher Education (MNiSW).

Appendix

The algorithm εDEag given in [2] (note that the algorithm has been directly quoted from the original paper):

Step 1 Initialization of an archive. Initial M individuals A={x_i, i=1, 2,…, M} are generated randomly in search space S and form an archive.
Step 2 Initialization of the ε level. An initial ε level is given by the ε level control function $\varepsilon(0)$.
Step 3 Initialization of a population. Top N individuals are selected from the archive A and form a population P={x_i}. The ranks of individuals are defined by the ε level comparison.
Step 4 Termination condition. If the number of function evaluations exceeds the maximum number of evaluation FEmax, the algorithm is terminated.
Step 5 DE operations (at most twice). Each individual x_i is selected as a parent. If all individuals are selected, go to Step 7. DE/rand/1/exp operation is applied and a new child x child is generated. A fixed scaling factor F0 and a fixed crossover rate CR0 are used with probability 0.95, and the randomly generated scaling factor F and the crossover rate CR0 are used with the probability 0.05. The third vector x^{p3} is selected from the archive A with probability 0.95 and selected from the population P with probability 0.05. If the new one is better than the parent based on the ε level comparison, the parent xi is immediately replaced by the trial vector xchild and go to Step 6. In the εDEag, not discrete generation

model but continuous generation model is adopted. Otherwise, the same operation is applied to the parent only once again.

Step 6 Gradient-based mutation. If $x$child is not feasible, or $\varphi(x\text{child}) > 0$, the child is changed by the gradient-based mutation with probability Pg until the number of the changes becomes Rg or $x$child becomes feasible. Go back to Step5 and the next individual is selected as a parent.

Step 7 Control of the ε level. The ε level is updated by the ε level control function ε(t).

Step 8 Go back to Step4.

References

[1] Z. Michalewicz, M. Schoenauer, Evolutionary algorithms for constrained parameter optimization problems,, Evol. Comput. 4 (1996) 1–32.

[2] T. Takahama, S. Sakai, Constrained optimization by the ε constrained differential evolution with an archive and gradient-based mutation, in: Proceedings of the Congress on Evolutionary Computation, 2010, pp. 1–9.

[3] J. Liang, S. Zhigang, L. Zhihui, Coevolutionary comprehensive learning particle swarm optimizer, in: Proceedings of the Congress on Evolutionary Computation, 2010, pp. 1–8.

[4] R. Eberhart, J. Kennedy, A new optimizer using particle swarm theory, in: Proceedings of the International Symposium on Micro Machine and Human Science, 1995, pp. 39–43.

[5] D.E. Goldberg, Genetic ALgorithms in Search, Optimization, and Machine Learning, Addison-Wesley Pub. Co., the University of Michigan, 1989 (Reading, Mass).

[6] R. Storn, K. Price, Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces,, J. Glob. Optim. 11 (1997) 341–359.

[7] N. Hansen, The CMA evolution strategy: a comparing review, Towards New Evolut. Comput. (2006) 75–102.

[8] J.C. Gilbert, J. Nocedal, Global convergence properties of conjugate gradient methods for optimization,, SIAM J. Optim. 2 (1992) 21–42.

[9] G. Dantzig, Linear Programming and Extensions, Princeton University Press, New Jersey, 1998.

[10] Z. Michalewicz, A survey of constraint handling techniques in evolutionary computation methods, Evolut. Program. IV (1995) 135–155.

[11] M.R. Bonyadi, Z. Michalewicz, Locating potentially disjoint feasible regions of a search space with a particle swarm optimizer, in: K. Deb (Ed.), Evolutionary Constrained Optimization, Springer-Verlag, To appear, 2014.

[12] Z. Michalewicz, D.B. Fogel, How to Solve It: Modern Heuristics, Springer-Verlag Inc., New York, 2004.

[13] J. Kennedy, R. Eberhart, Particle swarm optimization, in: Proceedings of the International Conference on Neural Networks, 1995, pp. 1942–1948.

[14] R. Mendes, J. Kennedy, J. Neves, The fully informed particle swarm: simpler, maybe better,, IEEE Trans. Evolut. Comput. 8 (2004) 204–210.

[15] M.A. Montes de Oca, T. Stützle, M. Birattari, M. Dorigo, Frankenstein's PSO: a composite particle swarm optimization algorithm, IEEE Trans. Evolut. Comput. 13 (2009) 1120–1132.

[16] M. Clerc, Particle Swarm Optimization, Wiley ISTE Ltd, UK, 2006.

[17] Y. Shi, R. Eberhart, A modified particle swarm optimizer, World Congr. Comput. Intell. (1998) 69–73.

[18] R. Poli, J. Kennedy, T. Blackwell, Particle swarm optimization an overview,, Swarm Intell. 1 (2007) 33–57.

[19] M. Clerc, J. Kennedy, The particle swarm – explosion, stability, and convergence in a multidimensional complex space, IEEE Trans. Evolut. Comput. 6 (2002) 58–73.

[20] I.C. Trelea, The particle swarm optimization algorithm: convergence analysis and parameter selection,, Inf. Process. Lett. 85 (2003) 317–325.

[21] F. van den Bergh, A. Engelbrecht, A new locally convergent particle swarm optimiser, Syst. Man Cybern. (2002) 96–101.

[22] F. Van den Bergh, A.P. Engelbrecht, A convergence proof for the particle swarm optimiser, Fundam. Inf. 105 (2010) 341–374.

[23] H. Wang, H. Sun, C. Li, S. Rahnamayan, J.-s. Pan, Diversity enhanced particle swarm optimization with neighborhood search,, Inf. Sci. 223 (2013) 119–135.

[24] M.M. Noel, A new gradient based particle swarm optimization algorithm for accurate computation of global minimum,, Appl. Soft Comput. 12 (2012) 353–359.

[25] D.N. Wilke, S. Kok, A.A. Groenwold, Comparison of linear and classical velocity update rules in particle swarm optimization: notes on diversity,, Int. J. Numer. Methods Eng. 70 (2007) 962–984.

[26] W.M. Spears, D.T. Green, D.F. Spears, Biases in particle swarm optimization,, Int. J. Swarm Intell. Res. 1 (2010) 34–57.

[27] X. Hu, R. Eberhart, Solving constrained nonlinear optimization problems with particle swarm optimization, World Multiconf. Syst. Cybern. Inf. (2002) 203–206.

[28] K.E. Parsopoulos, M.N. Vrahatis, Particle swarm optimization method for constrained optimization problems,, Intell. Technol. – Theory Appl.: New Trends Intell. Technol. 76 (2002) 214–220.

[29] G. Coath, S.K. Halgamuge, A comparison of constraint-handling methods for the application of particle swarm optimization to constrained nonlinear optimization problems, in: Proceedings of the Congress on Evolutionary Computation, vol. 4, 2003, pp. 2419–2425.

[30] X. Hu, R.C. Eberhart, Y. Shi, Engineering optimization with particle swarm,, Swarm Intell. Symp. (2003) 53–57.

[31] U. Paquet, A.P. Engelbrecht, A new particle swarm optimiser for linearly constrained optimisation, in: Proceedings of the Congress on Evolutionary Computation, 2003, pp. 227–233.

[32] G.T. Pulido, C.A.C. Coello, A constraint-handling mechanism for particle swarm optimization, in: Proceedings of the Congress on Evolutionary Computation, 2004, pp. 1396–1403.

[33] T. Takahama, S. Sakai, Constrained optimization by ε constrained particle swarm optimizer with ε-level control, Soft Comput. Transdiscip. Sci. Technol. (2005) 1019–1029.

[34] J. Liang, P. Suganthan, Dynamic multi-swarm particle swarm optimizer with a novel constraint-handling mechanism, in: Proceedings of the Congress on Evolutionary Computation, 2006, pp. 9–16.

[35] T. Takahama, S. Sakai, Solving constrained optimization problems by the ε constrained particle swarm optimizer with adaptive velocity limit control, IEEE Conf. Cybern. Intell. Syst. (2006) 1–7.

[36] T. Takahama, S. Sakai, Constrained optimization by the ε constrained differential evolution with gradient-based mutation and feasible elites, in: Proceedings of the Congress on Evolutionary Computation, 2006, pp. 1–8.

[37] Q. He, L. Wang, An effective co-evolutionary particle swarm optimization for constrained engineering design problems,, Eng. Appl. Artif. Intell. 20 (2007) 89–99.

[38] U. Paquet, A.P. Engelbrecht, Particle swarms for linearly constrained optimisation,, Fundam. Inf. 76 (2007) 147–170.

[39] R. Brits, A.P. Engelbrecht, F. Van den Bergh, A niching particle swarm optimizer, in: Proceedings of the 4th Asia-Pacific Conference on Simulated Evolution And Learning, 2002, pp. 692–696.

[40] A. Engelbrecht, B. Masiye, G. Pampard, Niching ability of basic particle swarm optimization algorithms, Swarm Intell. Symp. (2005) 397–400.

[41] R. Brits, A. Engelbrecht, F. Van den Bergh, Locating multiple optima using particle swarm optimization, Appl. Math. Comput. 189 (2007) 1859–1883.

[42] X.D. Li, Niching without niching parameters: particle swarm optimization using a ring topology, IEEE Trans. Evolut. Comput. 14 (2010) 150–169 (Aug).

[43] N. Hansen, A. Ostermeier, Adapting arbitrary normal mutation distributions in evolution strategies: the covariance matrix adaptation, in: Proceedings of the Congress on Evolutionary Computation, 1996, pp. 312–317.

[44] R. Mallipeddi, P. Suganthan, Problem definitions and evaluation criteria for the CEC 2010 Competition on Constrained Real-Parameter Optimization, Nanyang Technological University, Singapore, Technical Report, 2010.

[45] M.R. Bonyadi, X. Li, Z. Michalewicz, A hybrid particle swarm with velocity mutation for constraint optimization problems, in: Genetic and Evolutionary Computation Conference, Amsterdam, The Netherlands, 2013, pp. 1–8.

[46] S.M. Elsayed, R.A. Sarker, D.L. Essam, Multi-operator based evolutionary algorithms for solving constrained optimization problems, Comput. Op. Res. 38 (2011) 1877–1896.

[47] Y. Shi, R. Eberhart, Parameter selection in particle swarm optimization, Evolutionary Programming VII (1998) 591–600.

[48] S. Helwig, R. Wanka, Particle swarm optimization in high-dimensional bounded search spaces,, Swarm Intell. Symp. (2007) 198–205.

[49] F. Van den Bergh, A.P. Engelbrecht, A study of particle swarm optimization particle trajectories, Inf. Sci. 176 (2006) 937–971.

[50] R. Poli, Mean and variance of the sampling distribution of particle swarm optimizers during stagnation, IEEE Trans. Evolut. Comput. 13 (2009) 712–721.

[51] M. Clerc, Stagnation Analysis in Particle Swarm Optimisation or What Happens When Nothing Happens, 2006.

[52] M.-Y. Cheng, K.-Y. Huang, H.-M. Chen, Dynamic guiding particle swarm optimization with embedded chaotic search for solving multidimensional problems, Optim. Lett. 6 (2011) 719–729.

[53] Z. Xinchao, A perturbed particle swarm algorithm for numerical optimization,, Appl. Soft Comput. 10 (2010) 119–124.

[54] J. Kennedy, R. Mendes, Population structure and particle swarm performance, in: Proceedings of the Congress on Evolutionary Computation, 2002, pp. 1671–1676.

[55] R. Mendes, Population topologies and their influence in particle swarm performance, Universidade do Minho, 2004.

[56] J. Liang, P.N. Suganthan, Dynamic multi-swarm particle swarm optimizer, in: Proceedings 2005 IEEE Swarm Intelligence Symposium, SIS 2005, pp. 124–129.

[57] H.M. Emara, Adaptive clubs-based particle swarm optimization, in: American Control Conference 2009, ACC'09, 2009, pp. 5628–5634.

[58] S.M. Elsayed, R.A. Sarker, D.L. Essam, Memetic multi-topology particle swarm optimizer for constrained optimization, in: Proceedings of the IEEE Congress on Evolutionary Computation, 2012, pp. 1–8.

[59] M. Newman, A.-L. Barabasi, D.J. Watts, The Structure and Dynamics of Networks, Princeton University Press, New Jersey, 2006.

[60] Y.-j. Gong and J. Zhang, Small-world particle swarm optimization with topology adaptation, in: Proceeding of the Fifteenth Annual Conference on Genetic and Evolutionary Computation Conference, 2013, pp. 25–32.

[61] Y. Cooren, M. Clerc, P. Siarry, Performance evaluation of TRIBES, an adaptive particle swarm optimization algorithm, Swarm Intell. 3 (2009) 149–178.

[62] S.W. Mahfoud, Niching methods for genetic algorithms, Urbana 51 (1995) 61801.

[63] Z. Michalewicz, C.Z. Janikow, GENOCOP: a genetic algorithm for numerical optimization problems with linear constraints,, Commun. ACM 39 (1996) 175.

[64] K. Deb, An efficient constraint handling method for genetic algorithms, Comput. Methods Appl. Mech. Eng. 186 (2000) 311–338.

[65] S.L. Campbell, C.D. Meyer, Generalized inverses of linear transformations, Soc. Ind. Math. (2009).
[66] C.A. Coello Coello, Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: a survey of the state of the art, Comput. Methods Appl. Mech. Eng. 191 (2002) 1245–1287.
[67] E. Mezura-Montes, C.A. Coello Coello, Constraint-handling in nature-inspired numerical optimization: past present and future, Swarm Evolut. Comput. 1 (2011) 173–194.
[68] D.V. Arnold, N. Hansen, A $(1+1)$-CMA-ES for constrained optimisation, in: Proceedings of the Fourteenth International Conference on Genetic and Evolutionary Computation Conference, 2012, pp. 297–304.
[69] M.R. Bonyadi, Z. Michalewicz, On the edge of feasibility: a case study of the particle swarm optimizer, in: Proceedings of the Congress on Evolutionary Computation, Beijing, China, 2014.
[70] M. Vrahatis, G. Androulakis, G. Manoussakis, A new unconstrained optimization method for imprecise function and gradient values,, J. Math. Anal. Appl. 197 (1996) 586–607.