Enhancing performance of particle swarm optimization through an algorithmic link with genetic algorithms

Citation: Deb, Kalyanmoy, and Nikhil Padhye. "Enhancing Performance of Particle Swarm Optimization through an Algorithmic Link with Genetic Algorithms." Computational Optimization and Applications 57.3 (2014): 761–794.

As Published: http://dx.doi.org/10.1007/s10589-013-9605-0

Publisher: Springer US

Persistent URL: http://hdl.handle.net/1721.1/103318

Version: Author's final manuscript: final author's manuscript post peer review, without publisher's formatting or copy editing

Computational Optimization and Applications manuscript No.
(will be inserted by the editor)

Enhancing Performance of Particle Swarm Optimization Through an Algorithmic Link with Genetic Algorithms

Kalyanmoy Deb · Nikhil Padhye

Received: date / Accepted: date

Abstract Evolutionary Algorithms (EAs) are emerging as competitive and reliable techniques for several optimization tasks. Juxtapositioning their higher-level and implicit correspondence; it is provocative to query if one optimization algorithm can benefit from another by studying underlying similarities and dissimilarities. This paper establishes a clear and fundamental *algorithmic linking* between particle swarm optimization (PSO) algorithm and genetic algorithms (GAs). Specifically, we select the task of solving unimodal optimization problems, and demonstrate that key algorithmic features of an effective Generalized Generation Gap based Genetic Algorithm can be introduced into the PSO by leveraging this algorithmic linking while significantly enhance the PSO's performance. However, the goal of this paper is not to solve unimodal problems, neither is to demonstrate that the modified PSO algorithm resembles a GA, but to highlight the concept of algorithmic linking in an attempt towards designing efficient optimization algorithms. We intend to emphasize that the evolutionary and other optimization researchers should direct more efforts in establishing equivalence between different genetic, evolutionary and other nature-inspired or non-traditional algorithms. In addition to achieving performance gains, such an exercise shall deepen the understanding and scope of various operators from different paradigms in Evolutionary Computation (EC) and other optimization methods.

Keywords Particle swarm optimization · genetic algorithms · real-parameter optimization · unified algorithms · algorithmic linking

Kalyanmoy Deb
Department of Electrical and Computer Engineering
Michigan State University
East Lansing, MI 48824, USA
E-mail: kdeb@egr.msu.edu
`http://www.egr.msu.edu/~kdeb`
Nikhil Padhye
Department of Mechanical Engineering
Massachusetts Institute of Technology
Cambridge, MA 02139, USA
E-mail: npdhye@mit.edu

1 Introduction

Classical optimization algorithms are mainly classified into two categories – (i) gradient based algorithms and (ii) direct search based algorithms [42, 40, 12]. Although both these types of algorithms use a single solution point in each iteration, there exist a number of population based classical optimization algorithms, such as Box's algorithm [7], adaptive random search methods [30], etc. Recent decades have marked developments in several non-traditional optimization methods such as simulated annealing [28], evolutionary algorithms [23, 18], particle swarm optimization [27], and other nature or bio-inspired algorithms. The emergence of several optimization paradigms provokes following basic questions: How evolutionary algorithms differ from one another in terms of their fundamental construct? Equally and importantly, is there an *algorithmic linking* between these algorithms? If so, can the desired properties of one algorithm can be transmitted over to another effectively? A recent study has proved that the performance of a differential evolution (DE) algorithm can be significantly enhanced by adopting a unified approach and borrowing operations from a real-parameter genetic algorithm (GA) [34].

Genetic algorithms were first suggested by John Holland in early sixties [22]. Thereafter in 1975 De Jong showed that GAs can be used as function optimizers [14]. Today, GAs constitute one of the unarguably dominant optimization methods. GAs are population based search approaches in which an initial population of solutions gets updated by three main operators: (i) *selection* – that chooses better solutions from the population, (ii) *recombination* – that produces *offspring* solutions by combining the selected solutions, and (iii) *mutation* – that alters the offsprings one at a time with a goal of maintaining diversity in the population. Over the past years, convergence properties of GAs have also been studied [44], along with numerous customizations for different search and optimization tasks. Similar developments have occurred in Evolutionary Programming (EP), Evolutionary Strategies (ES), Genetic Programming (GP), etc., however, they are not central focus of this paper.

In 1995 [27], a new paradigm in nature-inspired meta-heuristics named particle swarm optimization (PSO) was introduced by simulating the behavior of organisms like those in the bird flocks or fish schools. In the original proposal, authors tied the roots of PSO to artificial life (A-life) and related it to GAs and evolutionary programming methods [17] based on their structural similarities. Over the past decade, PSO has gained a wide-spread popularity within research communities mainly due to its simplistic implementation, reported success on benchmark test problems, and acceptable performance on application problems. Based on the canonical form of original PSO, majority of research focus in the past has been spent on improving the optimizer's performance in various contexts. In the original paper [27], the developers recognized that PSO's child update procedure is similar to a GA's crossover operator. Another study [15] discussed a resemblance of PSO with an evolutionary algorithm; and even considered PSO as an evolutionary algorithm. Importantly, they recognized that each operator of an EA is somehow present in a PSO, though not directly. They went on to describe that PSO does not have a direct crossover operator, but the concept of a crossover is also present in the PSO. Furthermore, a PSO does not have an explicit selection operator, but its use of global best and personal best solutions in child creation acts as inherent selection operators. Other studies have also looked at the similarities and differ-

ences between EA and PSO, namely [1], and characterized an EA procedure as primarily a competitive evolutionary process; whereas PSO as a cooperative one.

In spite of such studies, we argue that one of the striking features of a PSO is that its child creation operator uses an *individualistic* approach in a sense that each population member uses its past state, current state and its personal best to create a new solution. The only information it shares with the other population members is the global best solution (if a fully-connected topology is assumed). In comparison, a GA allows its recombination operator to be applied among any two or more population members, and relies its search on its recombinative aspect involving multiple population members (*populistic approach* for creating new and hopefully better solutions [18].

Although qualitative similarities exist between PSO and a GA, most of the PSO approaches use the standard particle update rule (we shall refer to this as child creation rule). Despite the original developer's realizations of PSO's similarity with other evolutionary algorithms there are is no systematic or a revealing study which develops an algorithmic link between a PSO and a GA. If a direct algorithmic link can be established, then PSO researchers can borrow knowledge from GAs to enhance the performance of the PSO algorithm and vice versa.

In this paper, we illustrate the advantage of building an algorithmic link between a PSO and a GA in the context of solving a particular class of unimodal problems — the problems having one or very few optima. For the unimodal problems, an efficient algorithm, when initialized far away from the optimum, must possess adequate diversity preserving and simultaneously convergence properties. This is important to avoid stagnation and to maintain sufficient exploration capability in order to drive the search towards the optimum.

A real-parameter genetic algorithm (GA) was developed to efficiently handle unimodal problems in the recent past [13] and reported a competitive number of overall function evaluations needed to find a near-optimal solution in comparison to a number of other evolutionary algorithms, including CMA-ES [20], evolution strategies [45], differential evolution [48], and a classical optimization method. In this study, we consider the same set of test problems and compare the performance of existing PSO algorithms to the best-known results. Simulations reveal that standard PSO implementations with commonly used parameter settings fail to perform well. When faced with such a predicament, the existing PSO framework does not provide a simple way forward. Instead of resorting to sophisticated options (such as multiple swarms, adaptive change in parameters etc.), we attempt to design a GA which is algorithmically identical to the basic PSO approach and then borrow effective GA operators to PSO while preserving its individualistic traits. Such a task helps to enhance the PSO performance significantly.

Our study here should not be misunderstood to say that if there already exists a GA to solve the problems of our interest, why do we care solving them by some other optimization paradigm? Neither it should dismissed based on the fact that we have focused on unimodal problems instead of more complex optimization problems. The study should not also be viewed as a process of making one algorithm similar to the other, rather the lack or presence of specific operators and the process of inclusion of operators from one algorithm to the other should be viewed as reflections of flexibilities and adaptibilities associated with algorithms – a matter which will make our understanding of different algorithms better and richer. Since every algorithm is likely to have a niche where it stands out as a pre-

ferred candidate over the other algorithm, we do not advocate abandoning one for the other based on the performance on certain set of problems. Rather, the goal in this paper is to emphasize the importance of establishing an algorithmic linking among different optimization algorithms, so that, as and when needed, ideas from one can be borrowed to enhance the performance of the other. As observed in this paper, while the ideas from GAs help to improve the performance of canonical PSO, on one of the test problems the performance of our enhanced PSO surpasses that of the best known GAs. We argue that the procedure adopted in this study is generic and can be applied to develop efficient and collaborative algorithms for solving other difficult optimization problems. In addition, such tasks should provide researchers with useful insights and understanding of different algorithmic aspects needed in an efficient optimization method.

In the remainder of this paper, we begin by providing a brief discussion on salient PSO studies. Thereafter, we present a genetic algorithm framework which is essentially identical to a canonical PSO algorithm in terms of its working principle. This is followed by a section in which we describe three unimodal test problems and the best performance results from an earlier study. PSOs with standard parameter settings are then applied to these problems[1]. Thereafter, additional operators from the existing GA study are borrowed one by one and judiciously added to the PSO algorithm in order to maintain PSO's individualistic characteristics. Once the final algorithm is developed, a detailed parametric study is performed to fine tune the PSO algorithm. The performance of the proposed PSO algorithm is then tested on large-dimensional (up to 500-variable) problems. Finally, the conclusions are provided and the importance of this study is highlighted.

2 Related PSO Studies and Algorithm Architecture

Popular studies in PSO have been focused on the task of single objective optimization. However, the swarm intelligence idea has been successfully applied to multi-modal optimization [4], multi-objective optimization [35,11], and several real-world applications. Few attempts have also been directed towards the development of theoretical models for particle trajectories in PSO. The proposed models have been validated empirically and often been found useful in explaining the swarm behavior, for example, see [10,26,32]. A detailed and complete recollection of past work is beyond the interest of this paper. For an extensive overview of past work in PSO, readers are referred to a recent survey [3]. For reviews on multi-objective optimization, multi-modal optimization, dynamic optimization, readers are referred to [43,4,6]. Next, we shall visit some well known extensions of PSO which have shown superior performance compared to the canonical PSO.

The canonical form of PSO consists of a population of particles, known as a swarm, with each member of the swarm being associated with a position vector $\mathbf{x}^t$ and a velocity vector $\mathbf{v}^t$. The size of these vectors is equal to the dimension of the search space. The term velocity ($\mathbf{v}^t$) at any iteration t indicates the directional distance that the particle has covered in the $(t-1)$-th iteration. Hence, this term can also be called as a *history* term. The velocity of the population members moving

[1] The source codes adopted in this paper are available at [33], or by e-mailing npdhye@gmail.com.

through the search space is calculated by assigning stochastic weights to $\mathbf{v}^t$ and the attractions from a particle's personal-best or 'pbest' ($\mathbf{p}_l$) and swarm's best or 'gbest' ($\mathbf{p}_g$), and computing their resultant vector. The 'pbest' vector indicates the best position attained by a particle so far, whereas the 'gbest' vector indicates the best location found so far in the entire swarm (this study implements popularly used fully informed swarm topology, that is, each particle knows the *best* location in the entire swarm rather than in any defined neighborhood).

The following equations describe the velocity and position update for i-th particle at any iteration t (we refer to this as the *child creation rule*):

$$\mathbf{v}_i^{t+1} = w\mathbf{v}_i^t + c_1\mathbf{r}_1 \ .* \ (\mathbf{p}_{l,i}^t - \mathbf{x}_i^t) + c_2\mathbf{r}_2 \ .* \ (\mathbf{p}_g^t - \mathbf{x}_i^t), \tag{1}$$

$$\mathbf{x}_i^{t+1} = \mathbf{x}_i^t + \mathbf{v}_i^{t+1}. \tag{2}$$

Here, $\mathbf{r}_1$ and $\mathbf{r}_2$ are random vectors (with each component in $[0, 1]$), and w, c_1 and c_2 are pre-specified constants. The $.*$ symbol signifies component-wise multiplication of two vectors. In each iteration, every particle in the population is updated serially according to the above position and velocity rules.

A little thought will reveal that the above child creation rule involves a population member $\mathbf{x}_i$ at the current generation, its position vector in the previous generation, and its best position so far. We argue that these vectors are all *individualistic* entities of a population member. The only non-individualistic entity used in the above child creation is the position of the globally best solution $\mathbf{p}_g$, which solely justifies the population aspect of the PSO. In this study, we treat the individualistic aspect as a trademark feature of PSO algorithm and argue that this is one of major differences of child creation rule between a PSO and a GA.

The parametric study on coefficients (w, c_1, and c_2) for terms $\mathbf{v}_i^t$, $(\mathbf{p}_{l,i}^t - \mathbf{x}_i^t)$ and $(\mathbf{p}_g - \mathbf{x}_i^t)$, respectively, were conducted in [9,47] and empirical studies revealed that in Equation (2), the w value should be about 0.7 to 0.8, and c_1 and c_2 around 1.5 to 1.7. Irrespective of the choice of w, c_1 and c_2, while working with the bounded spaces the velocity expression often causes particles to 'fly-out' of the search space. To control this problem, a velocity clamping mechanism was suggested in [16]. The clamping mechanism restricted the velocity component to lie in $[-v_{\max,j} \ v_{\max,j}]$ along each dimension (j). Usually, $v_{\max,j}$ along j-th dimension is taken as 0.1 to 1.0 times the maximum value of x_i along the i-th dimension. Such a clamping mechanism does not necessarily ensure that newly created particles shall remain in the search space. However, since we are concentrating on solving unconstrained problems in this study, a velocity restriction mechanism is not needed.

The velocity term ($\mathbf{v}_i^t$) indicates a particle's ability to explore the search space while it moves under the attraction of 'pbest' and 'gbest' points. In initial phases of the search, a wide exploration is favored; whereas towards the end, a more focused search is desired. To achieve this, the concept of decreasing inertia weight (w) was introduced in [46]. The strategy has gained popularity in promoting convergence. The idea of varying coefficients was also extended successfully to dynamically update the parameters c_1 and c_2 in [41,49].

Premature convergence has been a major issue in PSO. The particles often accelerate towards the swarm's best location and the population collapses. A study classified under swarm stability and explosion [10] proposed a velocity update rule based on 'constriction factor' (χ). In the presence of χ, the velocity update

Equation 1 becomes:

$$\mathbf{v}_i^{t+1} = \chi \left(\mathbf{v}_i^t + c_1 \mathbf{r}_1 \ .* \ (\mathbf{p}_{l,i}^t - \mathbf{x}_i^t) + c_2 \mathbf{r}_2 \ .* \ (\mathbf{p}_g^t - \mathbf{x}_i^t) \right). \quad (3)$$

Equation 3, is one of the most popular versions of the velocity update rule in PSO studies.

Another related topic of interest is PSO's hybridization with evolutionary or other methods, such as genetic algorithms (GAs), or differential evolution (DE) and ant colony optimization (ACO). Although hybridizations are considered as deviation from the social metaphor of a swarm, as the notion of 'pbest' and 'gbest' is given lesser attention in such studies, but many instances of hybridization have reported favorable results. Representative works in this direction are briefly summarized as follows: In [2], the tournament selection was applied to replace the velocity and position of poorly performing particles with those of good performers. NichPSO algorithm was developed by training the swarm using Kennedy's cognition only model [25] and then creating artificial niches. The approach was successful in showing better convergence. In [29], a sub-population approach was borrowed to improve PSO's convergence. The sub-population approach was particularly beneficial in multi-modal problems but suffered on efficiency in unimodal problem. In [50], PSO and DE were used in tandem to develop 'DEPSO'. Here, DE and canonical PSO operators were employed at alternate generations. DEPSO showed an improvement on certain test problems with higher dimensionality. In [21], a Gaussian mutation was combined with velocity update rule to improve PSO's performance. In [24], the upper half of best performing members were regarded as elites and treated as a separate swarm. The lower half was evolved with genetic crossover and mutation operators. This approach was shown to outperform GA and PSO. In [38], authors replaced the particle update rules by using a quadratic crossover operator to generate new solutions. The performance of their approach was shown to be significantly better than the canonical PSO.

Despite some efforts in recognizing the similarity of PSOs and GAs and their need for hybridization, a fundamental question remains: 'How are these two algorithms related to each other so that, if needed, some salient features of one can be introduced to the other in order to enhance the latter's performance?' In this paper, we attempt to understand their similarity and differences by presenting an equivalent GA for a standard PSO algorithm. Once such an equivalence is established, knowledge of one algorithm can be efficiently transferred to the other. Without such algorithmic linking trial and error modification attempts may be futile. In the following section, we draw one such an algorithmic linking and subsequently illustrate how such a connection can be utilized to improve the performance of PSO algorithm.

3 Algorithmic Linking Between PSO and GAs

Both PSO and GAs are population-based optimization procedures. As already mentioned elsewhere [27], the child creation rule involving $\mathbf{p}_l$ and $\mathbf{p}_g$ in the particle swarm optimizer is conceptually similar to the recombination operation used in the GA. We illustrate this aspect by first outlining the essentials of a GA procedure.

First, a GA works with a population of solutions in each iteration. Second, an explicit selection operator chooses a few good solutions from the population based

on objective and constraint functions. Third, present and/or past good solutions (called parents) are utilized to create new (called children) solutions by using essentially two types of operations: recombination and mutation. Although these terms are borrowed from natural genetics and natural selection; in the algorithmic context each operator has a specific role. Following the semantics of natural genetics the recombination and mutation operations can be understood as follows:

Recombination operator: Each recombination event involves *more than one* evolving population members (parents) in creating a new solution (child). An evolving population member is one that is not fixed throughout a run, but is updated with generations. The recombination operation can be an exchange of partial information among participating parents, or a blending of parent entities, or any other operation (or a series of operations) that involves information from two or more evolving solutions directly or indirectly. In this sense, if the population mean solution is used in the child creation process, it is a recombination operator, since the determination of the population mean involves more than one evolving population members and it is an evolving entity.

Mutation operator: Each mutation event involves *one and only one* evolving population member in creating a new solution. According to this principle, if an update on a current population member involves a fixed variable vector (constant throughout the GA run, say the best population member of the initial population), it is still a mutation operation, despite the use of two solutions in the process. If a current population member is updated by a point-by-point hill-climbing method or a classical gradient-based optimization method, it is still a mutation operation, despite the o creation of many intermediate solutions along the way. It is important to note that these intermediate solutions are derived from a single (current) population member and not using any other member from any present or past evolving GA populations.

The above demarcation of recombination and mutation operators makes a clear distinction of their working principles and can be utilized in developing or evaluating new recombination and mutation operators for solving a new class of problems.

After the new population (or a subpopulation) is created through selection, recombination and mutation operations, it is usually compared with the current (and previous) population and good solutions are preserved for the next generation. This is known as an *elite-preservation* operation. Often an external population (commonly known as an *archive*) is maintained and updated at the end of each generation; to keep a record and utilize previously-found best solutions in subsequent generations. One or more termination criteria are usually employed to decide the completion of a GA run.

Based on the above general working principle, let us now consider a specific GA algorithm with a population P^t and an archive population A^t of size N at generation t. In the initial generation, the archive population is identical to the initial GA population, that is, the i-th population member and its partner in the archive (the i-th archive member) are identical: $A_i^0 = P_i^0$ for $i = 1, 2, \ldots, N$. We call the i-th archive member as $\mathbf{p}_{l,i}^t$. The best archive member at generation t is declared as the globally-best solution ($\mathbf{p}_g^t$), since it preserves the best-so-far member at every population slot. In order to create a new (i-th) child solution, we choose to use three parent solutions for this GA: (i) the population member itself, $\mathbf{x}_i^t$, (ii) its corresponding archive partner, $\mathbf{p}_{l,i}^t$, and (iii) always the current

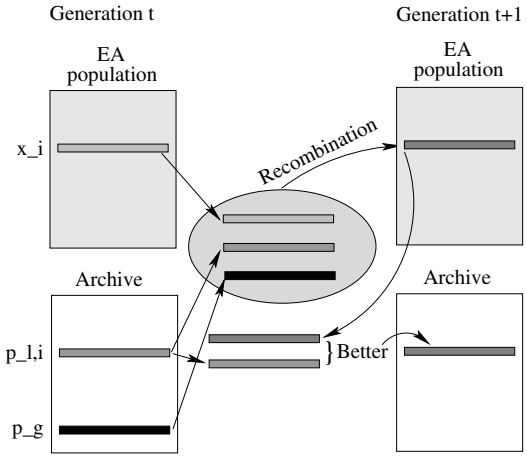

Fig. 1: An archive based GA formulation having a recombination operator involving three evolving solutions.

globally best archive member, $\mathbf{p}_g^t$. The following child creation rule is used for this purpose:

$$\mathbf{x}_i^{t+1} = \mathbf{x}_i^t + c_1\mathbf{r}_1 \ .* \ \left(\mathbf{p}_{l,i}^t - \mathbf{x}_i^t\right) + c_2\mathbf{r}_2 \ .* \ \left(\mathbf{p}_g^t - \mathbf{x}_i^t\right). \tag{4}$$

In the above equation, c_1 and c_2 are user-specified constants, $\mathbf{r}_1$ and $\mathbf{r}_2$ are random vectors whose components lie in [0,1]. The new solution is now compared with the corresponding archive member $\mathbf{p}_{l,i}^t$ and the better of the two replaces i-th archive member. Figure 1, illustrates this child creation.

GA population members (along with their archive partners and the $\mathbf{p}_g^t$) are selected serially one-by-one for the child creation process. After creating the new population and updating the archive members, the globally best archive solution for the next generation ($\mathbf{p}_g^{t+1}$) is set as the best member in the archive population. The procedure is continued iteratively till a termination criteria is met.

The GA described above follows a generational model and resembles an elitist evolutionary optimization algorithm which assigns a deterministic and equal selection pressure to each population member. Due to the use of three evolving population and archive members in the child creation process described by Equation 4, this operation qualifies as a recombination operator. There is no specific mutation operation introduced in the above GA procedure. The archive population gets updated along with the GA population, but no explicit GA operations are performed independently on the archive population.

A little thought will reveal that the procedure described above is fundamentally a standard PSO algorithm without the velocity term. We suggest a modification to the above archive-based GA procedure a little later to establish an exact algorithmic link with the velocity-based PSO algorithm. But the similarity of above archive-based GA with PSO minus the velocity term is striking. In fact, both algorithms though look different at the outset, are identical at their cores.

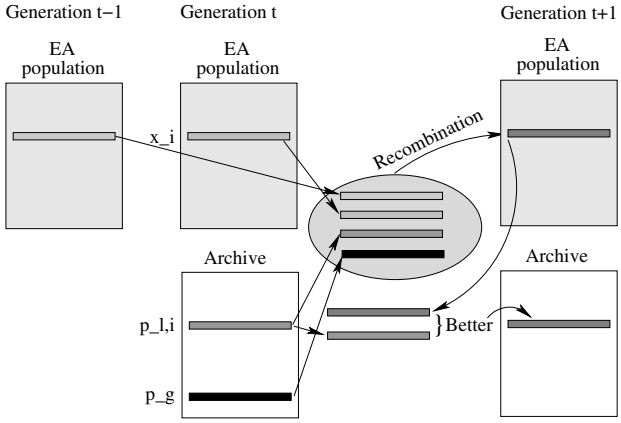

Fig. 2: A sketch of an GA having a recombination operator involving four evolving solutions.

Usually, in a GA the creation of the new population solely depends on the current population members (and sometimes the current archive members). Thus, conventionally a GA is also a Markov chain. However, a GA necessarily need not always be a Markov chain and one can design a GA which uses multiple past populations in creating the offspring population. In this spirit, we modify the above GA by including i-th population member from the $(t-1)$-th generation in its recombination operation, as follows:

$$\mathbf{x}_i^{t+1} = \mathbf{x}_i^t + w\left(\mathbf{x}_i^t - \mathbf{x}_i^{t-1}\right) + c_1\mathbf{r}_1 \ .* \ \left(\mathbf{p}_{l,i}^t - \mathbf{x}_i^t\right) + c_2\mathbf{r}_2 \ .* \ \left(\mathbf{p}_g^t - \mathbf{x}_i^t\right). \qquad (5)$$

Figure 2 shows a sketch of the procedure in which four solutions are involved in the recombination process. Although this is an uncommon recombination operation (due to the use of $\mathbf{x}_i^{t-1}$), but it definitely qualifies as a recombination operation due to the use of multiple existing evolving solutions as parents.

Often in PSO studies, an additional mutation operation is performed on $\mathbf{x}_i^{t+1}$ [21]. Such an operation is similar to the standard mutation operation employed after the recombination in the GA.

The above discussions clearly indicate that for the standard PSO algorithm, there exists an equivalent GA which is algorithmically identical to the PSO algorithm. Next, we discuss the advantages of establishing such an algorithmic link.

3.1 Advantages of Linking Two Algorithms

Let us consider a scenario, in which we are interested in enhancing the performance of a particular PSO algorithm by modifying its operators. The standard practice in this direction is to perform a parametric study on w, c_1 and c_2. Other performance enhancing methods include the use of non-uniform probability distributions for $\mathbf{r}_1$ and $\mathbf{r}_2$ as well [9,26] or the use of multiple swarms [31]. But most modifications use the same child creation rule given in Equation 5. If the update rule is strictly

followed, then we may be restricting its potential as an optimization algorithm for solving a wide-range of problems. A clear understanding of each term in the PSO child creation rule and their equivalence with an algorithmically linked GA opens up new ways of performing child creation in PSO. Embracing the equivalence with an GA procedure allows us to employ several other strategies and operations which have been developed since the inception of GAs and other evolutionary algorithms, such as evolution strategies and evolutionary programming.

The discussions so far indicate that the dominant difference between the PSO and GA child creation processes lies in the use of the history term (the velocity term $w(\mathbf{x}_i^t - \mathbf{x}_i^{t-1})$). We speculate that the PSO can benefit from the vast GA literature, and by noting the similarities and fundamental differences between the two algorithms ideas can be borrowed from one algorithm to enhance the performance of the other.

4 PSO and Unimodal Optimization

In this study, we consider unimodal problems (having one optimum solution) or problems having a few optimal solutions, so as to test an algorithm's ability (i) to progress towards the optimal region from the supplied initial search space, and (ii) to focus and find the optimum with a pre-defined precision. A previous study [13] considered a number of evolutionary algorithms for these problems, such as a generalized generation gap (G3) model using a parent-centric crossover (PCX) operator, a differential evolution, a number of evolution strategies (ESs), the CMA-ES, and a classical gradient-based optimization method. Following test problems were used:

$$F_{\text{elp}} = \sum_{i=1}^{n} i x_i^2 \quad \text{(Ellipsoidal function)} \tag{6}$$

$$F_{\text{sch}} = \sum_{i=1}^{n} \left(\sum_{j=1}^{i} x_j \right)^2 \quad \text{(Schwefel's function)} \tag{7}$$

$$F_{\text{ros}} = \sum_{i=1}^{n-1} \left(100(x_i^2 - x_{i+1})^2 + (x_i - 1)^2 \right) \quad \text{(Generalized Rosenbrock's function)} \tag{8}$$

We use them in our study and initially consider $n = 20$ dimensional version of the problems. The first two problems have their minimum at $x_i^* = 0$ with $F^* = 0$ and the third function has its minimum at $x_i^* = 1$ with $F^* = 0$. In order to study an algorithm's ability to progress towards the optimal region, we initialize the population by restricting $x_i \in [-10, -5]$ for all i, in all the problems; however in the subsequent generations we do not confine the solutions to lie in the above range. Most of the PSO studies initialize their population randomly around the optimal solution (since the optimal solution is known a priori in test problems) and employ different strategies to enforce the solutions back into the feasible region by modifying particle velocities and/or positions. The boundary handling strategies used in such studies may provide an undue advantage to the algorithm and hence we exempt from using any such procedure [36] here. Initializing population around the known optimum region does not allow to test an algorithm's ability to approach

towards the optimal region, rather in most cases it only tests the algorithm's ability to focus on the optimal solution. In this study, we investigate both aspects of an optimization algorithm by considering two evaluation criteria.

First, a population is initialized away from the optimal region and we count the number of function evaluations needed for the algorithm to find a solution close to the optima and we call this our first evaluation criterion S_1. We arbitrarily choose a limiting objective function value of $F = 0.1$ for this purpose. Due to unimodality of the search space, this criterion will denote how fast an algorithm is able to reach the optimal region. The second evaluation criterion (S_2) involves the overall number of function evaluations needed to find a solution having a function value very close to the optimal function value. We choose $F = 10^{-20}$ for this purpose. The difference between these two criteria provides us with an idea of the algorithm's ability to approach to and also to focus near the true optimum.

4.1 An Effective Generalized Generation Gap based Genetic Algorithm (G3-PCX)

A previous study [13] proposed a steady-state GA which solved the above test problems in a computationally efficient manner. The G3-PCX algorithm was designed to solve unimodal problems and utilized a real-parameter based parent-centric recombination (PCX) operator. One iteration of algorithm is described as follows:

Step 1: From the population P, select the best solution as a parent and choose $\mu - 1$ other parents randomly.
Step 2: Generate λ offspring from the chosen μ parents using the PCX recombination scheme.
Step 3: Choose two parents p_a and p_b at random from the population P.
Step 4: From a combined subpopulation of two chosen parents (p_a and p_b) and λ created offspring, choose the best two solutions and replace the chosen two parents (p_a and p_b).

Summarily, the G3-PCX algorithm has following key properties:

- It is a steady-state algorithm. In each iteration, at most two new solutions are updated in the population. Instead of the newly created child solutions to wait until all new offspring members are created, they can be chosen as a parent right after their creation. For solving unimodal problems, this property provides a high selection pressure for above-average population members.
- It uses an elite-preserving operator as the children are compared with the parents before constructing the population for the next generation.
- It involves the globally best solution so far in every child creation process. For solving unimodal problems, it is an efficient operation.
- It uses a recombination operator that creates new solutions around the best solution. This strategy helps in quickly approaching the optimum solution for the unimodal problems.

Clearly G3-PCX algorithm is a greedy optimization algorithm specifically designed to solve unimodal problems efficiently. The earlier extensive study on G3-PCX algorithm reported the best, median and worst number of function evaluations

Table 1: G3-PCX results, as reported in [13].

	F_{elp}			F_{sch}			F_{ros}		
	Best	Median	Worst	Best	Median	Worst	Best	Median	Worst
S_2	5,744	6,624	7,372	14,643	16,326	17,712	14,847 (38)	22,368	25,797

needed based on 50 different runs on the 20-variable version of the three problems with the identical S_2 criterion. Table 1 presents these results. In a comparison with a CMA-ES [19,20], a couple of self-adaptive evolution strategies [45,5], differential evolution [48] and a quasi-Newton classical method [42], the function evaluations with G3-PCX reported in the above table were found to be the smallest. The F_{ros} problem has two local optima and G3-PCX was able to converge to the globally optimal solution in 38 out of 50 runs. Thus, we treat G3-PCX algorithm as the benchmark algorithm for solving the selected test problems here.

4.2 Standard PSO Algorithms

Several PSO settings are applied on these test problems during the course of this study and we evaluate their performances vis-a-vis our target results obtained using G3-PCX. To eliminate the random effects and gather results of statistical importance, every algorithm is tested on each problem 50 times (every run starting with a different initial population). A particular run is terminated if the evaluation criterion (S_1 or S_2 depending upon which one is chosen) is met, or the number of function evaluations exceed one million. If less than 50 runs are successful then we report the count of successful runs in brackets. In this case, the best, median and worst number of function evaluations of the successful runs are reported. In case none of the runs is successful i.e. neither S_1 nor S_2 is reached within a maximum of one million function evaluations, 'DNC' (Did Not Converge) label is placed. In 'DNC' cases, we report the best, median and worst attained function values of the best solution in 50 runs. To distinguish the unsuccessful results from successful ones, we present the fitness value information of the unsuccessful runs in italics.

First, we consider the application of a standard PSO algorithm using following commonly used parameter settings:

1. Equation 2 with ($w = 0.5$, $c_1 = 1.0$, $c_2 = 1.0$): This setting has been adopted from [37], and gives equal average weight to velocity term and attractions towards 'gbest' and 'pbest'.
2. Equation 2 with ($w = 0.7$, $c_1 = 1.47$, $c_2 = 1.47$): This setting has been adopted from [9] and reported to perform well based on several experimentations conducted by the authors.
3. Equation 2 with ($\bar{w}$, $c_1 = 1.47$, $c_2 = 1.47$): In this setting w is decreased linearly (denoted by $\bar{w}$) over the PSO iterations from an initial value of 0.7 to 0.0, as discussed in [9]. Linearly decreasing w promotes a wider search in initial phases and focused search towards the end.
4. Equation 3 with ($\chi = 0.729$, $c_1 = 2.05$, $c_2 = 2.05$): This modified PSO move prevents swarm explosion and ensures stability. The parameter settings have been taken from [10]. It should be noted that this setting ($\chi = 0.729$, $c_1 = c_2 =$

Table 2: Different PSO results on three problems. The numbers shown are globally-best function values obtained by the respective PSO.

	F_{elp}			F_{sch}			F_{ros}		
	Best	Median	Worst	Best	Median	Worst	Best	Median	Worst
PSO with $w = 0.5$, $c_1 = 1.0$ and $c_2 = 1.0$									
S_1	*1,683.0*	*2,577.5*	*3,709.4*	*3073.2*	*6,279.9*	*12,756.5*	*435,400.1*	*896,497.6*	*1,419,791.0*
	(DNC)	(DNC)	(DNC)	(DNC)	(DNC)	(DNC)	(DNC)	(DNC)	(DNC)
PSO with $w = 0.7$, $c_1 = 1.47$, and $c_2 = 1.47$									
S_1	7,600	8,800	11,700	20,200	26,000	33,200	49,400 (11)	338,400	997,700
S_2	43,700	47,600	51,900	163,100	177,400	192,500	*1.50e-11*	*3.99*	*11.53*
							(DNC)	(DNC)	(DNC)
PSO with constriction factor based update with $\chi = 0.729$, $c_1 = 2.05$ and $c_2 = 2.05$									
S_1	9,600	11,000	131,000	24,800	34,000	42,000	422,800 (39)	493,700	832,700
S_2	58,000	61,700	67,100	218,400	242,400	263,700	*3.05e-05*	*4.84e-04*	*3.99*
							(DNC)	(DNC)	(DNC)

2.05) is equivalent to PSO with $w = 0.729$ and $c_1 = c_2 = 1.49$ in the form of Equation 2.

The population size of swarm is set equal to 100 and kept fixed for the rest of this paper unless stated otherwise.

For the standard PSO with $w = 0.5$, $c_1 = 1.0$, and $c_2 = 1.0$, results are reported in Table 2. With this setting, PSO is unable to find a solution having a fitness value of 0.1 or less even after spending one million function evaluations in each of the 50 runs for all the problems. Since all runs are unsuccessful, we only report the best, median and worst of the best function values obtained by the PSO procedure over the 50 runs. The best function value for F_{elp} is $F = 1,683$, whereas the true optimum function value for this problem is zero. Similarly, for F_{sch} and F_{ros}, the corresponding obtained function values are much bigger than zero. These results indicate that when the standard PSO is set to evolve freely without any restrictions for e.g. variable bounds its search is ineffective.

Next, another experiment is conducted based on a more popular parameter setting with $w = 0.7$, $c_1 = 1.47$ and $c_2 = 1.47$ and results are shown in Table 2. The performance of the PSO with this setting is significantly better compared to the PSO with the previous parameter setting. For problems F_{elp} and F_{sch}, this PSO is successful in all 50 runs with both S_1 and S_2 criteria. Although this is a significant improvement, the values corresponding to S_2 criterion for both these problems are much worse than those obtained using the G3-PCX algorithm (Table 1). For the F_{ros} problem, PSO is successful with S_1 criterion only in 11 out of 50 runs and no success is obtained with the S_2 criterion. The best function value attained by this PSO is $1.50(10^{-11})$. Large parameter values allowed the PSO to spread its solutions faster towards the optimal region. The chosen parameter values make a good balance between progress towards the current best solutions and laying emphasis on history and making a remarkable convergence pattern.

Since the above parameter settings performed relatively well, we try a modification to the same setting by decreasing w from 0.7 to zero and keep c_1 and c_2 same as before ($= 1.47$). The best, median and worst results obtained are exactly the same as that obtained as those with $w = 0.7$ (Table 2).

As a final choice of PSO variants, the constriction factor based update rule for velocity update is tried next and the results are tabulated in Table 2. The results are comparable to those obtained with the previous parameter setting, requiring a slightly higher function evaluations. But here, the number of successful runs with the S_1 criterion for F_{ros} is better: 39 out of 50. Since the PSO works with an unbounded search space the χ factor is useful in preventing disproportionate increase in velocity. A closer investigation will reveal that a PSO move based on Equation 3 with $\chi = 0.729$, $c_1 = 2.05$ and $c_2 = 2.05$ is equivalent to a PSO move based on Equation 2 with $w = 0.729$, $c_1 = 1.49$ and $c_2 = 1.49$. The constriction based PSO lays slightly more emphasis on the history (or exploration) term. Due to an overall good performance in general on all three problems, in rest of this paper we shall base our modifications on PSO with the constriction factor based update rule.

The above four parameter settings were also tested with different population sizes and showed minor improvements in some occasions, but mostly resulted in a deterioration. For the sake of brevity, we do not include these results here. It is also worth mentioning that in [39] several useful techniques to enhance the performance of the PSO have been used. In particular, the adaptation of the PSO parameters using computational statistics was attempted and results were compared with the standard settings leading to marginal improvements. However, the reported accuracies were not close to the target error set in this study. Overall, the performance of the standard PSOs on the chosen test problems needs a significant improvement in order to match the results reported by the G3-PCX algorithm (Table 1).

Under current circumstances where PSOs with popularly used parameter settings of child creation rule and population sizes are unable to yield satisfactory results within defined number of function evaluations, what options does a PSO practitioner have to proceed further? The PSO literature offers a number of complicated options: (a) instead of using static values of c_1 and c_2 throughout a run, these parameters can be changed adaptively as a linear/quadratic/exponential function of the generation count, (b) rather than using a *uniform* distribution for c_1 and c_2 in $[0, 1]$ other distributions (may be a Gaussian) be tried, (c) use of multi-swarm, or memory swarms, or tribes approach, and a number of other modifications be employed [9]. These options have their own additional parameters which must be set right for the resulting algorithm to work well. For the dynamically changing parameters functional forms of variations are required. Other distributions like Gaussian require mean and variance values. The multi-swarm approaches require the neighborhood size, number of swarms, etc. Due to intricateness with these concepts and additional requirement of identifying correct parameters settings, the dynamics for the evolution of good solutions may not be obvious. Moreover, if the sophisticated implementations fail to work, there is no straight-forward way to find clues in order to set the parameters appropriately.

When faced with such a predicament, instead of going for complicated extensions, we resort to the algorithmic link between PSO and the specialized G3-PCX algorithm, and investigate which features are missing in the PSO.

```
If (RandomDouble(0.0,1.0)≤ t_f)
 For i=1:N
  If (particle.x_i≤=gbest.x_i)
   {
    Low=particle.x_i - δ*(gbest.x_i- particle.x_i);
    High=particle.x_i + δ*(gbest.x_i- particle.x_i);
   }
  Else
   {
    Low=particle.x_i - δ*(particle.x_i-gbest.x_i);
    High=particle.x_i + δ*( particle.x_i-gbest.x_i);
   }
  End
  particle.x_i=particle.x_i+
    RandomDouble(Low-particle.x_i, High-particle.x_i);
 End
End
```

Fig. 3: Proposed mutation operator for PSO.

5 Modifying PSO using Strategies from Genetic Algorithms

In this section, we borrow the algorithmic link between the archive-based GA and PSO discussed in Section 3, and import ideas from G3-PCX algorithm in order to enhance PSO's performance.

5.1 Using a Mutation Operator

First and foremost, we realize that mutation is a straight-forward concept which can be applied without much thought. The importance of mutation operator is discussed later in Section 5.1.1. Mutation operator has already been used by researchers in PSO [21]. The concept that resembles mutation in PSO was originally introduced as 'craziness'. Now this is popularly known as the 'turbulence factor' (having a parameter t_f). The goal of adding a turbulence factor in PSO is to promote diversity in the PSO population so as to avoid pre-mature convergence. In the past, *Random*, *Gaussian*, *Cauchy*, etc., distribution based mutations have been employed for this purpose [9] and often reported to improve the performance of PSO. Since, we are working with an unbounded search space, its appropriate to define the operating space for the mutation. We prescribe this to be the region between the particle and the 'gbest'. The details of the mutation operator are given in Figure 3. The proposed mutation operator considers a solution to be altered and calculates its position with respect to the global best solution along each dimension. Then, using a fraction (δ) of the distance along each dimension, new particle position is computed in the neighborhood of global best. This mechanism is expected to promote both, diversity and search, around the global best solution, which can be regarded desirable feature for the case of unimodal problems.

Table 3: The χ-PSO algorithm with mutation operator having $t_f = 0.25$.

	F_{elp}			F_{sch}			F_{ros}		
	Best	Median	Worst	Best	Median	Worst	Best	Median	Worst
S_1	7,400	8,500	10,200	19,500	258,000	31,900	53,200 (39)	177,900	243,000
S_2	42,100	45,200	48,600	159,200	170,300	184,600	*1.75e-18* (DNC)	*1.18e-12* (DNC)	*3.99* (DNC)
Best S_2 so far	**42,100**	**45,200**	**48,600**	**159,200**	**170,300**	**184,600**	*1.75e-18* (DNC)	*1.18e-12* (DNC)	*3.99* (DNC)

Table 3 shows the results with $t_f = 0.25$ and $\delta = 0.5$ (chosen based on some preliminary experiments and kept fixed, unless specified otherwise). It can be observed from the Table 3 that the resulting PSO performs better on F_{elp} and F_{sch} compared to PSO without mutation. For F_{ros} problem, median and worst function evaluations for the S_1 criterion are also better. However, the mutation-based PSO is still not able to reach our S_2 targetted accuracy of the order 10^{-20}. Still the best function value found in this case is better than the earlier PSO algorithms.

The final row in the table shows the best results obtained so far. Since the current mutation-based PSO has achieved better results compared to original PSO algorithms, we show these numbers in bold.

5.1.1 An Analysis Through a Diversity Measure

To further understand the effect of mutation operator, we compare the performances of two χ-PSO algorithms (one with $t_f = 0.0$ and another with $t_f = 0.25$) against the benchmark G3-PCX algorithm. This is done through studying the *Diversity Metric* defined as follows:

$$\textit{Diversity Metric} = \frac{\sum_{i=1}^{K} \sqrt{(\mathbf{x}^{(i)} - \mathbf{x}^c) \cdot (\mathbf{x}^{(i)} - \mathbf{x}^c)^T}}{K}, \tag{9}$$

where, $\mathbf{x}^c = \frac{\sum_{i=1}^{K} \mathbf{x}^{(i)}}{K}$, is the position vector of the centroid of the top K ($\leq N$) population members. In this study, we use $K = 0.5N$. Thus, the above metric indicates the diversity of top 50% population members in the variable space. The idea is to compute this metric periodically after a few function evaluations for all the three algorithms and compare their evolution trends.

The diversity metric plots for F_{elp} and F_{sch} for a typical simulation run are shown in Figures 4 and 5, respectively. To keep the comparison fair, we start with the same initial population for all the three algorithms. We plot these figures till a function value of 0.1 is achieved. The figures show that the diversity of the top half of the population with G3-PCX algorithm shows in initial rise and then rapidly falls down, indicating that the population first expands itself to locate the optimal region and thereafter the population diversity rapidly decreases to find the optimum solution with an accuracy of 0.1 in the optimal function value. Since, G3-PCX is successful in finding the optimum in a computationally fast manner, it is reasonable to assume that while solving the same problem by another population based algorithm the population diversity should also follow a similar trend.

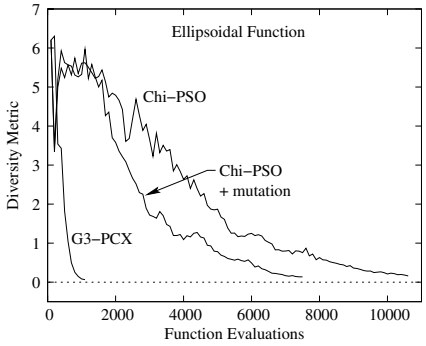

Fig. 4: Diversity metric for F_{elp} function for χ-PSO with $t_f = 0.0$, $t_f = 0.25$, and G3-PCX.

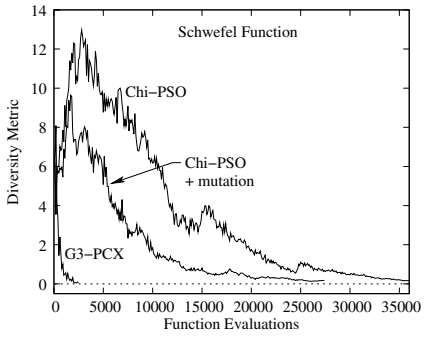

Fig. 5: Diversity metric for F_{sch} function for χ-PSO with $t_f = 0.0$, $t_f = 0.25$, and G3-PCX.

We observe that the evolution of the population diversity of χ-PSO is quite different and slow compared to that of G3-PCX. However, when the mutation operator is added, the diversity behavior starts approaching G3-PCX trend. Interestingly, all three algorithms have a similar behavior i.e. diversity increases in the beginning and then reduces, except that their rates of diversity rise and fall are different.

The function evaluations of PSO with mutation (Table 3) are also better compared to PSO without mutation (Table 2). We conclude that the trend of faster rise followed by a fall in the population diversity is a desired characteristic while solving unimodal problems. We also observed a similar outcome with F_{ros} (but due to space considerations do not show that plot).

After successful introduction of mutation operator in χ-PSO, we are now ready to add other GA operators one at a time, hopefully, to reduce the difference in performances between the resulting PSO and G3-PCX and also understand the missing operators in standard PSO needed to be efficient in solving unimodal problems. In all these efforts, we maintain the individuality aspect of PSO (discussed in Section 2).

5.2 Steady-State Update of Global Best Solution

In a standard PSO algorithm, the same global best solution is used for all N population members to create N child-solutions. If we think of the PSO procedure as an equivalent GA procedure as discussed in Section 3 and compare that GA procedure with the G3-PCX algorithm, we can immediately think of a steady-state PSO implementation; in which after every child is created, the global best solution is updated, that is, every newly created child is compared with the current global best solution and if the child is found to be better, then the child replaces the global best solution. With this *elite-preservation*, if a good child solution is created it can immediately start contributing to the creation of new solutions by participating in recombination with other population members. In solving unimodal (or likewise)

Table 4: Results using the χ-PSO algorithm with global solution update after creation of every child (steady-state approach) and mutation with $t_f = 0.25$.

	F_{elp}			F_{sch}			F_{ros}		
	Best	Median	Worst	Best	Median	Worst	Best	Median	Worst
S_1	6,900	8,200	9,700	18,900	25,800	33,500	353,000 (41)	217,800	265,000
S_2	42,300	43,700	46,900	146,600	164,800	178,600	*1.37e-17* (DNC)	*2.14e-10* (DNC)	*3.99* (DNC)
Best S_2 so far	**42,300**	**43,700**	**46,900**	**146,600**	**164,800**	**178,600**	*1.75e-18* (DNC)	*1.18e-12* (DNC)	*3.99* (DNC)

problems, such a steady-state approach, reportedly, has been useful; particularly the example of G3-PCX [13].

Table 4 shows the results obtained by using the steady-state and mutation-based PSO. An improvement in terms of function values is obtained for F_{elp} (in terms of median performance) and F_{sch}. The algorithm is still not able to solve F_{ros} with the desired accuracy.

Improved performance can be explained based on the fact that a better solution (if created by the child update process from any population member) does not have to wait at most N solution creations (meaning at most N function evaluations) to contribute to the search process. This has a domino effect in finding better solutions swiftly.

Importantly, this simple idea may not have surfaced easily if we only focused on the PSO child creation rule alone, as it is normally done. An equivalence of PSO with a GA framework and the importance of the steady-state operation in G3-PCX in solving similar problems allows us to modify the PSO algorithm to a steady-state PSO algorithm, and makes a significant improvement in the performance of the PSO.

We have also tested the performance of purely mutation driven steady-state PSOs by switching off the velocity and position updates and discovered worse performances. Thus, the mutation operator presented in this paper by itself is unable to carry out an adequate search and is effective only in presence of other PSO operations.

5.3 Pre-selection: Comparison of Parent and Child

The above modification is sufficiently interesting to look for other updates by which we can enhance the performance of the PSO algorithm. The G3-PCX algorithm replaces a randomly chosen solution from the population with the newly created child or the parent solution (whichever is better). In a PSO, every population member has a meaningful ancestor which is the corresponding population member from the previous generation. An update of a random population member will render the notion of ancestor meaningless. However, the concept of comparing a population member with its newly created child, and accepting the better of the two can be a good idea in general and is helpful in maintaining the population diversity (suggested as early as in 1970 in the context to GAs [8]).

Table 5: Results of mutation-based χ-PSO with steady-state and parent-child comparison approaches.

	F_{elp}			F_{sch}			F_{ros}		
	Best	Median	Worst	Best	Median	Worst	Best	Median	Worst
S_1	20,400	25,600	31,700	78,000	119,600	178,000	103,200 (26)	491,700	969,800
S_2	124,400	139,500	153,300	754,900	851,400	961,100	*2.35e-10* (DNC)	*5.08e-02* (DNC)	*4.99* (DNC)
Best S_2 so far	42,300	43,700	46,900	146,600	164,800	178,600	*1.75e-18* (DNC)	*1.18e-12* (DNC)	*3.99* (DNC)

Table 6: The χ-PSO algorithm with mutation $t_f = 0.25$ and the parent-child comparison approach.

	F_{elp}			F_{sch}			F_{ros}		
	Best	Median	Worst	Best	Median	Worst	Best	Median	Worst
S_1	18,700	23,500	31,000	66,900	106,500	170,500	197,600 (28)	710,000	989,300
S_2	127,600	136,200	146,600	719,000	818,000	970,900	*9.32e-09* (DNC)	*3.69e-02* (DNC)	*7.69* (DNC)
Best S_2 so far	42,300	43,700	46,900	146,600	164,800	178,600	*1.75e-18* (DNC)	*1.18e-12* (DNC)	*3.99* (DNC)

5.3.1 Steady-State and Parent-Child Comparison

First, we implement parent-child comparison (pre-selection) along with the steady-state approach on the mutation-based PSO. Table 5 shows that the results are not as good as the mutation-based PSO approach (Table 3). The performances are also poor compared to mutation-based PSO with steady-state approach (Table 4).

Overemphasis on newly created good solutions in conjunction with the steady-state approach introduces additional selection pressure for the current best population members. This causes a major degradation in the performance of the overall procedure.

So far, the best performance is observed with the addition of the steady-state approach and the mutation operator to the original χ-PSO algorithm (Table 4). Next, we investigate if the parent-child comparison with mutation alone (instead of steady-state update) provides a better balance between exploration and elite preservation.

5.3.2 Parent-Child Comparison Alone

The parent-child comparison is used with the mutation-based PSO here. Table 6 shows the results. The performance here is not much different from the previous approach and clearly the use of parent-child comparison as an elite enhancement is detrimental. However, as seen in Table 4, the steady-state approach alone with the mutation-based PSO is a better approach.

Table 7: Results of mutation-based steady-state χ-PSO with a random parent selection operation.

	F_{elp}			F_{sch}			F_{ros}		
	Best	Median	Worst	Best	Median	Worst	Best	Median	Worst
S_1	7,300	8,500	10,000	25,600	32,400	39,700	83,300 (35)	332,000	415,100
S_2	43,100	45,500	48,000	187,200	212,700	233,000	*3.13e-13* (DNC)	*1.85e-06* (DNC)	*3.99* (DNC)
Best S_2 so far	42,300	43,700	46,900	146,600	164,800	178,600	*1.75e-18* (DNC)	*1.18e-12* (DNC)	*3.99* (DNC)

5.4 Selection of Parents

Since the steady-state approach has made a positive improvement in the performance of the PSO algorithm, the order of selecting a parent for its child creation may be an important matter. If a better parent is operated earlier, the possibility of creating a good child early on is higher and in the presence of steady-state procedure; the performance of the overall algorithm may improve. In an evolutionary algorithm, good parents are usually selected for child generation by a separate *selection* operator. However, in the standard PSO the population members are selected serially. Thus, noting the principle of GAs, we introduce two selection operations one by one on the mutation-based steady-state PSO procedure as discussed previously in the subsection 5.2.

First, we introduce a random selection operation, in which a population member is chosen randomly for creating a new child solution. Although the population members are not expected to be in any order at any particular generation due to the random initial placement of solutions in the original PSO algorithm, the use of steady-state procedure may bias better solutions to be placed in some order after a sufficiently large number of generations. Thus, we believe that the use of a random selection procedure of parents may produce different results than without the use of this operator. Table 7 presents these results.

In a comparison with Table 4, this table shows that the inclusion of random selection does not improve the performance of the mutation based steady-state PSO approach on any test problem. However, the performance is comparable.

Next, we use the commonly-used binary tournament selection (without replacement) to select a parent solution. The PSO child creation rule is applied to the selected parent. Table 8 presents the results. An improvement in performance is obtained on F_{elp} by using the tournament selection operator. The performances on the other two problems do not show an improvement over the best results obtained so far.

So far, we have observed that the constriction based PSO update rule with an additional mutation operator, steady-state 'gbest' update, and tournament-based parent selection scheme is a good combination. Of the three problems, F_{elp} and F_{sch} problems are solved well, however the problem F_{ros} still remains to be unsolved with the specified accuracy. We now revisit the parent-child comparison operator along with the so far best designed PSO and make one final consideration of the parent-child comparison operator.

Table 8: Results of mutation-based steady-state χ-PSO with binary tournament selection operation.

	F_{elp}			F_{sch}			F_{ros}		
	Best	Median	Worst	Best	Median	Worst	Best	Median	Worst
S_1	5,700	6,700	7,900	22,000	29,700	37,500	134,200 (41)	262,200	290,100
S_2	34,600	36,600	38,400	169,900	191,200	217,600	*1.93e-13* (DNC)	*7.58e-09* (DNC)	*3.99* (DNC)
Best S_2 so far	**34,600**	**36,600**	**38,400**	146,600	164,800	178,600	*1.75e-18* (DNC)	*1.18e-12* (DNC)	*3.99* (DNC)

Table 9: Results of mutation-based steady state and parent-child χ-PSO with tournament selection.

	F_{elp}			F_{sch}			F_{ros}		
	Best	Median	Worst	Best	Median	Worst	Best	Median	Worst
S_1	9,300	12,800	16,400	50,600	67,900	97,100	42,500 (36)	292,400	557,300
S_2	67,900	75,100	80,700	400,600	460,300	542,700	*3.99e-16* (DNC)	*4.11e-07* (DNC)	*3.99* (DNC)
Best S_2 so far	34,600	36,600	38,400	146,600	164,800	178,600	*1.75e-18* (DNC)	*1.18e-12* (DNC)	*3.99* (DNC)

5.5 Mutation, Steady-state, Parent Selection and Parent-child Comparison

Table 9 shows that the combined approach with the parent-child comparison is not better than the combined approach without the parent-child comparison (Table 8). From these results, we conclude that the parent-child comparison, in general, is not a good approach with other features already present in the PSO algorithm. Thus, we do not continue with this operation any further.

5.6 Discussions on PSO Child Creation Rule

So far, we have used the standard PSO child creation rule with parameters obtained after a detailed study (as discussed in Section 4.2). Several GA operators are then introduced through the PSO-GA connection established in Section 3.

In Table 10, we compare the best PSO results found so far with those obtained by the G3-PCX procedure [13] in achieving a function value of 10^{-20} for all three problems. The problem F_{elp} performs the best with mutation, steady-state, tournament selection based PSO and the problem F_{sch} performs the best with mutation based PSO alone, though the former algorithm is not too far in terms of its performance. However, none of the PSO updates are able to solve F_{ros} to the desired accuracy.

The above table clearly shows that despite the enhancement in PSO's performance, the best PSO procedure still requires an order of magnitude more function evaluations compared to the G3-PCX algorithm. If the PSO child creation rule is to be strictly followed, we need to possibly now look for more sophisticated procedures suggested in the PSO literature, such as multiple swarms, memory swarms,

Table 10: Comparison of G3-PCX with best of PSO algorithms with PSO child creation rule on three test problems for S_2.

	F_{elp}			F_{sch}			F_{ros}		
	Best	Median	Worst	Best	Median	Worst	Best	Median	Worst
G3-PCX	**5,744**	**6,624**	**7,372**	**14,643**	**16,326**	**17,712**	**14,847 (38)**	**22,368**	**25,797**
Modified PSO	34,600	36,600	38,400	146,600	164,800	178,600	(DNC)	(DNC)	(DNC)

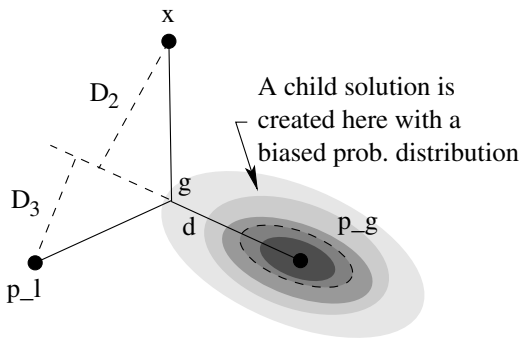

Fig. 6: For three solutions $\mathbf{x}$, $\mathbf{p}_l$ and $\mathbf{p}_g$, the region for creating a child solution is shown using the proposed PCX update rule.

etc. Since, we could improve the performance of PSO by understanding its equivalence with an GA procedure and borrowing GA operators to PSO (while using the PSO child creation rule), it is time we may borrow further algorithmic ideas from the GA literature and experiment by replacing PSO's child creation rule itself. In the next section, we perform this task using parent centric child creation operator from G3-PCX.

6 Modifying PSO Further with Different Solution Update Rules

The PCX operator requires three or more parent solutions and uses a parent-centric recombination in which the probability distribution is computed from the vector differences of the parents utilized in creation of child solutions [13]. The probability distribution is centered around the currently best parent solution. Figure 6 shows the working of the PCX operation on three parent solutions on a two-variable problem. The description of PCX is provided next.

The mean vector $\mathbf{g}$ of the chosen μ (=3 is used here) parents is first computed. For each child solution, one parent $\mathbf{x}^{(p)}$ $(= \mathbf{p}_g)$ is chosen. The direction vector $\mathbf{d}^{(p)} = \mathbf{x}^{(p)} - \mathbf{g}$ is then calculated. Thereafter, from each of the other $(\mu-1)$ parents, perpendicular distances D_i to the line $\mathbf{d}^{(p)}$ are computed and their average $\bar{D}$ is

computed. The child solution is then created as follows:

$$\mathbf{y} = \mathbf{x}^{(p)} + w_\zeta \|\mathbf{d}^{(p)}\| \mathbf{e}^{(p)} + \sum_{i=1,\ i\neq p}^{n} w_\eta \bar{D} \mathbf{e}^{(i)}, \tag{10}$$

where $\mathbf{e}^{(i)}$ are the n orthonormal bases that span the subspace. The direction $\mathbf{e}^{(p)}$ is along $\mathbf{d}^{(p)}$. The parameters w_ζ and w_η are zero-mean normally distributed variables with variance σ_ζ^2 and σ_η^2, respectively.

To borrow the PCX child creation operator into the PSO approach, we take help of our PSO-GA link discussed in Section 3. If we treat Equation 5 as a recombination process in which four evolving population members ($\mathbf{x}^{t-1}$, $\mathbf{x}^t$, $\mathbf{p}_l$ and $\mathbf{p}_g$) are *blended* together to create the child solution $\mathbf{x}^{t+1}$, we can think of other blending operations that are commonly used in the GA literature. Here, we borrow the blending idea from the PCX operator [13] involving three parents. Instead of using equation 5 to create a child solution $\mathbf{x}^{t+1}$, we use the global best solution $\mathbf{p}_g$, a population member's personal best solution $\mathbf{p}_l$, and the population member $\mathbf{x}$ itself, as three parents in the PCX operator. Later, we use all four parents in the PCX operator. In some sense, the PCX update rule uses a different probability distribution proportional to the differences between the parents ($\mathbf{p}_g$, $\mathbf{p}_l$ and $\mathbf{x}$) than that used in the usual PSO child creation rule. This modification in the child creation process still maintains the PSO's individualistic trait, as discussed before. Our effort in drawing the algorithmic link allows us to incorporate such changes at this stage where we have pretty much exhausted different PSO variants with various parameter settings and infusion of various evolutionary operators. We shall now investigate whether the performance difference between PSO and G3-PCX observed in Table 10 is due to the difference in the probability distribution of creating the child solutions.

6.1 Steady-State PSO with PCX Update Rule with No History

Here, we study the effect of replacing the PSO child creation rule with the PCX update rule. In the PCX operation child solutions are created around the globally best solution $\mathbf{p}_g$ and therefore there is a sufficient exploration potential. Whereas using the PSO child update, solutions are only created in the space spanned by the three vectors (i.e. velocity, attraction towards personal best, and attraction towards the global best).

First, we do not consider the velocity term, that is, $w = 0$. Although there is no direct way to compute velocity here, but we can still estimate velocity based on the displacement that occurred over the previous iteration as $\mathbf{v}^t = (\mathbf{x}^{t+1} - \mathbf{x}^t)$. Based on the findings of the previous sections we adopt mutation and steady-state operations which were also the two most effective strategies. The two PCX parameters are taken as $w_\zeta = 0.17$ and $w_\eta = 0.17$ based on some preliminary experiments. The results for this case are shown in Table 11. The table indicates that obtained solutions are not better than those presented in Table 10. These results probably can be explained as follows. In the unimodal problems, the individual population members are expected to improve in a continual manner and the personal-best solution $\mathbf{p}_l$ is most likely to be identical to the population member ($\mathbf{x}$) itself. When this happens, the PCX operator produces a child solution along

Table 11: The χ-PSO algorithm with PCX update, mutation ($t_f = 0.25$), steady state, and $w = 0$. Individual, Gbest and Pbest are three parents.

	F_{elp}			F_{sch}			F_{ros}		
	Best	Median	Worst	Best	Median	Worst	Best	Median	Worst
S_1	18,200	33,600	53,200	19200.0	29,800	45,300	296,200 (19)	673,900	978,400
S_2	134,000	170,500	211,600	157,700	184,800	201,900	*1.59e-04* (DNC)	*3.28e-01* (DNC)	*1.12* (DNC)
Best S_2 so far	34,600	36,600	38,400	146,600	164,800	178,600	*1.75e-18* (DNC)	*1.18e-12* (DNC)	*3.99* (DNC)

Table 12: The χ-PSO algorithm with distinct parent based PCX update rule, mutation ($t_f = 0.25$), and steady-state approach.

	F_{elp}			F_{sch}			F_{ros}		
	Best	Median	Worst	Best	Median	Worst	Best	Median	Worst
S_1	1,200	1,400	1,700	3,200	5,400	8,800	12,000 (42)	20,500	24,100
S_2	5,600	6,200	6,600	29,800	33,900	40,600	61,100 (42)	80,600	93,400
Best S_2 so far	**5,600**	**6,200**	**6,600**	**29,000**	**33,900**	**40,600**	**61,100 (42)**	**80,600**	**93,400**

the line joining $\mathbf{x}$ and the global best solution $\mathbf{p}_g$ (this corresponds to the case when D_2 and D_3 are zero in Figure 6). Such a line search may not be able to maintain adequate diversity for the algorithm to constitute an effective search in a 20-dimensional space. Thus, along with the PCX update rule, we may need to introduce additional diversity.

We tested three mechanisms for this purpose: (i) inclusion of the velocity term, (ii) increase of mutation probability, and (iii) choosing three *distinct* members as parent solutions. The updates (i) and (ii) failed to improve the performance of the resulting algorithms to the desired expectation, but the third approach makes a significant improvement, as discussed next.

6.2 Steady-State PSO with PCX based Update Rule with Distinct Parents

In this approach, we first check three solutions ($\mathbf{x}^t$, $\mathbf{p}_l$ and $\mathbf{p}_g$) for their candidacy as parents. If they are distinct, we use them in the PCX child creation rule. If they are not distinct, we find the three distinct solutions of the four solutions ($\mathbf{x}^{t-1}$, $\mathbf{x}^t$, $\mathbf{p}_l$ and $\mathbf{p}_g$) and use them as parents. If only two distinct solutions are present among the above four solutions, we then perform a PCX operation with $\bar{D} = 0$. On the other hand, if all solutions are identical, we do not perform the PCX update at all (rather update the solutions based on velocity alone). Table 12 shows the obtained results. The table shows a remarkable change in the performance of the resulting algorithm. Now, all three problems are solved to the desired accuracy and the required function evaluations are in the same ball-park as that obtained by G3-PCX. In fact, for F_{elp}, our proposed algorithm shows a better performance than previous G3-PCX results. Since all three parents for PCX are now distinct

(in most occasions), the PCX operator creates children solutions all around $\mathbf{p}_g$ and like in the G3-PCX operator, our proposed algorithm finds the right balance between diversity preservation and convergence for its progress and convergence towards the optimum.

To gain a sense of similarity, we again study the plots of the 'Diversity metric', given in equation (9) for PCX-PSO and G3-PCX on the same initial population. The plots for all three functions are shown in Figures 7, 8, and 9. The diversity plots show a high degree of resemblance for the two optimizers. The corresponding diversity metric value for the original χ-based PSO is also plotted in the figures as well. From these figures we observe that proposed modifications have enhanced the original χ-PSO approach to a level where not only the performances are very similar to G3-PCX but evolution of population is also similar based on the plots.

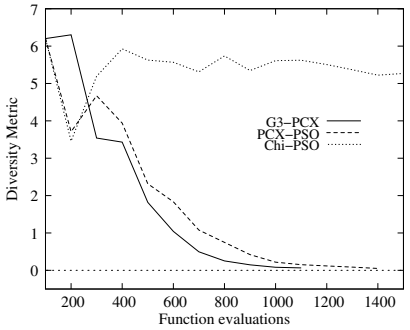

Fig. 7: Diversity metric values for G3-PCX and PCX-PSO are similar for F_{elp}.

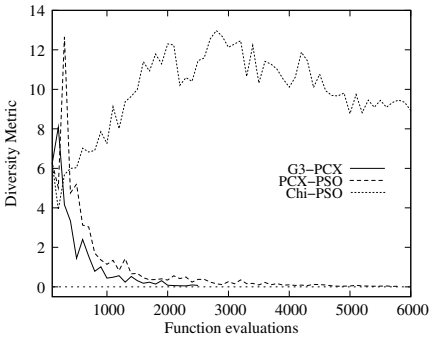

Fig. 8: Diversity metric values for G3-PCX and PCX-PSO are similar for F_{sch}.

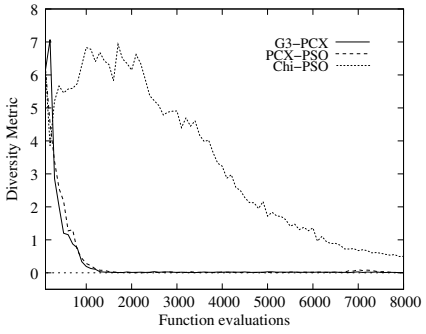

Fig. 9: Diversity metric values for G3-PCX and PCX-PSO are similar for F_{ros}.

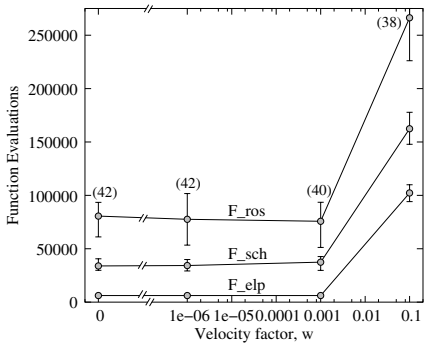

Fig. 10: Parametric study with velocity factor w.

There is a drastic change in the behavior of the modified PSO algorithm, evident from the dynamics of the population diversity with generation. Importantly, if we did not draw the algorithmic link between the PSO algorithm and GA, it would have been difficult to think of such a modification. It is also clear from the performance of the PSO that its standard child creation rule was not as efficient for solving unimodal problems as it is done with the PCX operator. The study performed here discovered systematic improvements on the canonical PSO in order to improve its performance.

6.3 Parametric Study

In the above PSO with PCX update, values for parameters such as t_f (equal to 0.25), population size (equal to 100) and w (equal to zero) were chosen based on some preliminary experiments. Next, we perform a detailed parametric study for these three parameters in order to find the best performance of the modified PSO approach.

First, we perform a parametric study on w by keeping $t_f = 0.25$ and $N = 100$. Figure 10 indicates that a w value close to zero performs better on all three problems. Any value within $w \in [0, 10^{-6}]$ performs almost equally well (including $w = 0.0$). These results indicate that the history term has little importance for the proposed algorithm in solving the unimodal problems. However, a small w may improve the performance occasionally.

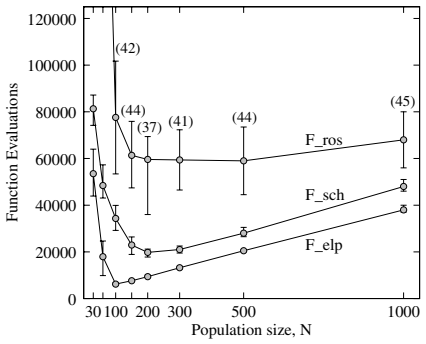

Fig. 11: Parametric study with population size.

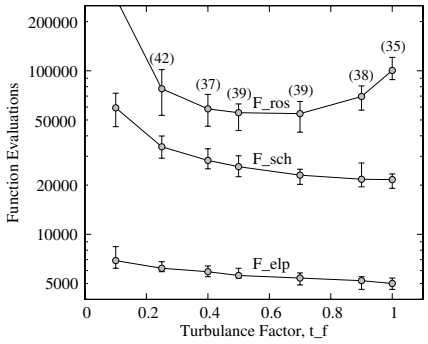

Fig. 12: Parametric study with mutation factor t_f.

Next, we fix $w = 10^{-6}$ and $t_f = 0.25$ and vary the population size N. Figure 11 shows the obtained results. The problem F_{elp} is quite sensitive to the population size and works well for $N = 100$ or so. However, for F_{sch} and F_{ros} problems, our modified PSO performs well around $N = 150$ and $N = 500$, respectively. Small population sizes do not provide adequate diversity for the algorithm to work at its best. Interestingly, a large population size is also found to be detrimental, as discovered in other PSO studies [10] as well.

Table 13: Comparison of G3-PCX with the best of modified PSO algorithms of this study for 20-variable problems.

	F_{elp}			F_{sch}			F_{ros}		
	Best	Median	Worst	Best	Median	Worst	Best	Median	Worst
G3-PCX	5,744	6,624	7,372	**14,643**	16,326	17,712	**14,847** (38)	22,368	25,797
Modified PSO	**4,600**	5,000	5,400	17,800	19,800	21,200	36,000 **(44)**	59,600	69,400

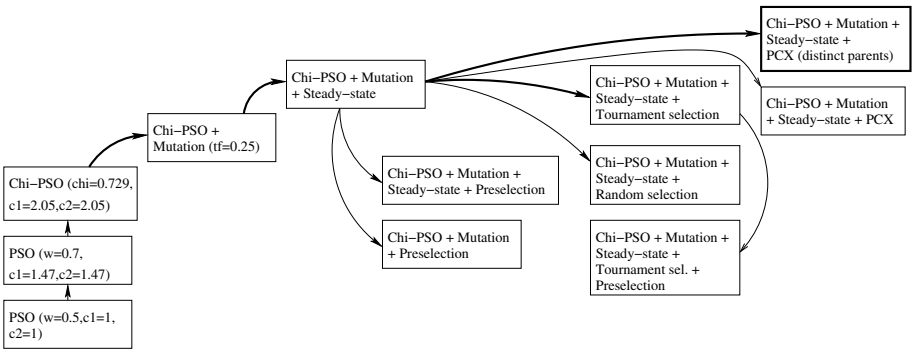

Fig. 13: The flowchart of the PSO updates studied here.

Finally, we fix $w = 10^{-6}$ and $N = 100$ and vary the mutation probability t_f. Figure 12 shows the results on all three problems. It is clear from the plot that the optimal t_f value depends on the problem. For F_{elp} and F_{sch} problems, the larger the value of t_f, the better is the performance. For F_{ros}, t_f around 0.5 to 0.75 performs well. It is quite evident that the proposed methodology requires a certain amount of mutation for good performance.

6.4 Final Comparison on 20-Variable Problems

Table 13 compares the best parametric results of the modified PSO approach with the G3-PCX results.

Importantly, from a standard PSO algorithm (Table 2), we are able to systematically modify the PSO algorithm by comparing and borrowing operations from an equivalent GA to develop a methodology which performs competitively. Figure 13 summarizes the flowchart of different updates tried in this study and their outcome in solving the problems. If an update has resulted in an improvement of performance it is shown with a bold arrowed line.

Interestingly, the resulting procedure is different from G3-PCX and it still follows a fundamental property of a PSO algorithm in that it maintains individuality of population members. A population member in the modified PSO algorithm still creates its child solution by using its immediate past solution, its personal-best solution, and population's global best solution – all the essential properties which uniquely characterizes a PSO algorithm. However, the child creation rule used in

Table 14: Parameters used for the scale-up study for all three problems.

n	N	t_f	$w_\eta = w_\zeta$
10	50	0.50	0.25
20	100	0.40	0.17
50	150	0.35	0.12
100	200	0.25	0.08
150	250	0.20	0.05
200	300	0.10	0.05
500	500	0.08	0.05

the modified approach is different from the usual PSO update rule. It uses a biased distribution around the global best solution rather than using a more uniform distribution used in a standard PSO study. Besides, the use of a steady-state update of the global best solution and a mutation operator are motivated by studying efficient genetic algorithms for solving similar problems in the past.

It is interesting also to note that the updates borrowed from G3-PCX into the PSO framework have outperformed the G3-PCX algorithm on F_{elp} problem. Compared to a median of 6,624 function evaluations over 50 runs (reported before), the modified PSO requires only 5,000 function evaluations (a 24% improvement). Developers of G3-PCX (including the first author of this study) could not foresee the algorithmic features outlined in this study and the algorithmic linking of G3-PCX, GA and PCX. The systematic experimentation have provided us with all such knowledge about solving unimodal problems that were not known before.

7 Solving Higher-dimensional Problems

Motivated by the development of a successful PSO-based algorithm for 20-variable problems, we now investigate its performance for solving higher-dimensional version of the same problems. We consider a number of variable sizes: 10, 20, 50, 100, 150, and 200. For F_{elp} and F_{sch}, we also try 500-variable versions. For each test problem, the best, median and worst number of function evaluations over 10 runs, so as to meet a termination criterion of $F = 10^{-10}$ from the true optimal objective value (zero) are noted. As the dimension (n) of a problem is increased, we use a larger population size (N) – a typical trend followed by EC researchers. We also use a smaller mutation probability (t_f) and smaller values of w_η and w_ζ, listed in Table 14. It is important to mention that we have not made an attempt to find optimal setting of these parameters, rather used values based on some test simulations.

The scale-up performances for three problems are shown in Figure 14. It is clear that the PCX enabled PSO algorithm is robust in the sense that the variation in the obtained solution over 10 independent runs is small. Importantly for problems F_{elp} and F_{sch}, the algorithm is able to converge in all 10 runs. Problem F_{ros} is reportedly a more difficult problem and the number of successful runs (out of 10 runs) are mentioned in the figure in brackets. These results amply demonstrate

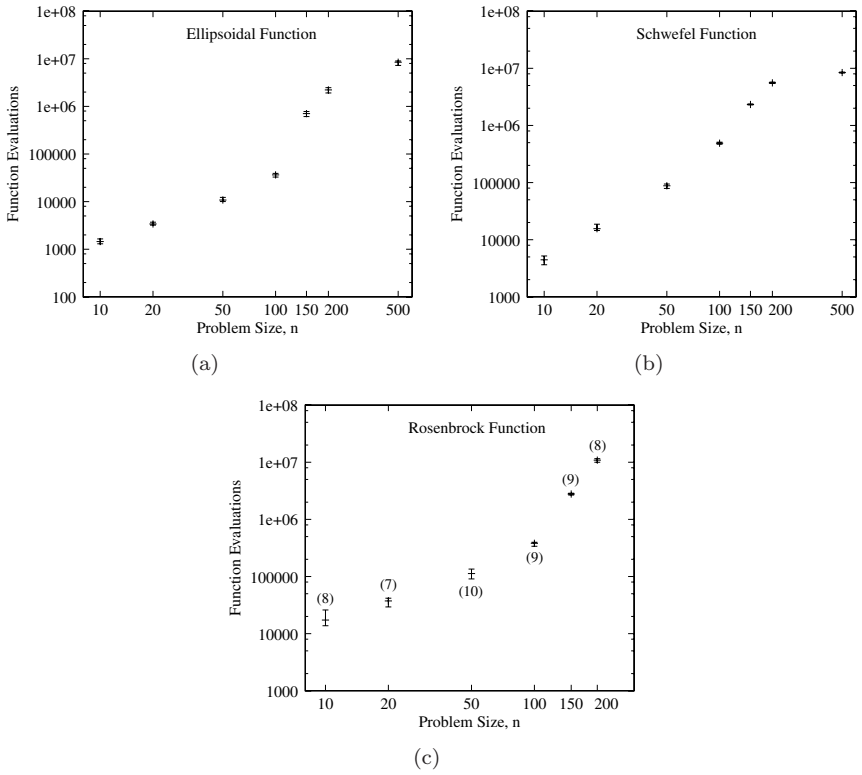

Fig. 14: Scale-up performances of PCX-PSO on 10 to 500 variables (a) $F_{\rm elp}$, all 10 runs in each case are successful. (b) $F_{\rm sch}$, all 10 runs in each case are successful. (c) $F_{\rm ros}$, the numbers in brackets show the successful runs in 10 runs.

the developed modified PSO algorithm's ability to find a near-optimum solution to large-dimensional version of the problems considered in this study.

8 Conclusions

Particle swarm optimization (PSO) algorithms are being practiced since 1995 for real-parameter optimization problems. The basic principle behind PSO is the individualistic approach and child creation rule which only use a particle's current location, particle's best location and population's best location. The PSO system is usually inferred as particles flying in the search space by combining social and cognition models. However, the essential idea is to utilize information from fellow population members and create better solutions.

In this paper, we have established a fundamental equivalence between PSO and a recombinative archive-based genetic algorithm. The algorithmic linking has allowed us to borrow useful operations from a previously reported efficient algorithm so as to enhance the performance of the PSO algorithm in consideration. We have chosen three 20-variable unimodal test problems from the literature. Stan-

dard PSOs with a number of commonly used parameter settings were unable to solve the problems to the desired accuracy. Thereafter, we have introduced several evolutionary operators within PSO by borrowing ideas from a computationally efficient genetic algorithm through the established algorithmic linking for solving the same problems. The use of additional mutation operator, steady-state update, and a binary tournament for parent selection on the constriction based PSO has emerged as the best strategy for solving the chosen problems. Although, the results obtained this far are significantly better than the standard PSO, yet an order of magnitude worse than the benchmark performance of G3-PCX.

After several modifications and investigations, we have realized that PSO's child creation rule was a major bottleneck in improving its performance any further. Next, we have modified the child creation step effectively, and introduced a parent-centric operator such that individualistic character of the PSO is retained. The usual entities, such as the population member, its personal best solution, the globally best solution and member's immediate past solution, all, have been used in the PCX update rule. Furthermore, the PSO's performance has been enhanced by using the steady-state approach and an additional mutation operator. The resulting PCX-PSO approach showed a competitive performance against a previously-reported efficient G3-PCX algorithm. A sensitivity study of parameters associated with the PCX-based PSO has demonstrated the robustness of the proposed approach. Finally, a scale-up study on as large as 500-variable problems has revealed that the proposed PCX-PSO algorithm is able to solve large-dimensional problems efficiently.

In summary, we emphasize here that one optimization algorithm can benefit from another algorithm by understanding an algorithmic linking between them and then borrowing important operators from one to the other. Although individual operations and their implementations in two algorithms may be different, the fundamental working principles and their contribution to the complete search process are important to understand. In this paper, we have shown this aspect between PSO and a specific GA in the context of solving unimodal problems. This study can be extended to perform collaborative algorithm developments for several other optimization tasks, such as multi-modal optimization, multi-objective optimization, dynamic optimization, combinatorial optimization, just to name a few. Furthermore, with the current frame-work in, possibly, the theoretical models of GAs can also adopted to study the behavior of PSO or other algorithms. Such research should give insights into the fundamental working principles of different algorithms, and identify the key challenges in the area of algorithm development.

Acknowledgments

Second author acknowledges discussions with Professor C.K. Mohan during his visits at Syracuse University on Particle Swarm Optimization.

References

1. Angeline, P.J.: Evolutionary optimization versus particle swarm optimization: Philosophy and performance differences. In: Proceedings of the 7th International Conference on Evolutionary Programming 7, pp. 601–610 (1998)

2. Angeline, P.J.: Using selection to improve particle swarm optimization. In: Proceedings of IEEE Congress on Evolutionary Computation (1998)
3. Banks, A., Vincet, J., Anyakoha, C.: A review of particle swarm optimization. part i: background and development. Natural Computing **6(4)**, 467–484 (2007)
4. Barrera, J., Coello, C.A.C.: A review of Particle Swarm Optimization Methods Used for Multimodal Optimization, vol. 248, pp. 9–37. Springer (2009)
5. Beyer, H.G.: Toward a theory of evolution strategies: Self-adaptation. Evolutionary Computation Journal **3**(3), 311–347 (1995)
6. Blackwell, T., Branke, J.: Multiswarms, exclusion, and anti-convergence in dynamic environments. IEEE Transactions on Evolutionary Computation **10(4)**, 459–472 (2006)
7. Box, M.J.: A new method of constrained optimization and a comparison with other methods. Computer Journal **8**(1), 42–52 (1965)
8. Cavicchio, D.J.: Adaptive search using simulated evolution. Ph.D. thesis, Ann Arbor, MI: University of Michigan (1970)
9. Clerc, M.: Particle Swarm Optimization. ISTE Ltd, UK/USA (2006)
10. Clerc, M., Kennedy, J.: The particle swarm - explosion, stability, and convergence in a multidimensional complex space. Evolutionary Computation, IEEE Transactions on **6**(1), 58–73 (2002)
11. Coello, C.A.C., Lechuga, M.S.: MOPSO: A proposal for multiple objective particle swarm optimization. In: Congress on Evolutionary Computation, pages 825-830. IEEE (2002)
12. Deb, K.: Optimization for Engineering Design: Algorithms and Examples. New Delhi: Prentice-Hall (1995)
13. Deb, K., Anand, A., Joshi, D.: A computationally efficient evolutionary algorithm for real-parameter optimization. Evolutionary Computation Journal **10**(4), 371–395 (2002)
14. DeJong, K.A.: An analysis of the behavior of a class of genetic adaptive systems. Ph.D. thesis, Ann Arbor, MI: University of Michigan (1975). Dissertation Abstracts International 36(10), 5140B (University Microfilms No. 76-9381)
15. Eberhart, R., Shi, Y.: Comparison between genetic algorithms and particle swarm optimization. In: Proceedings of the Seventh Annual Conference on Evolutionary Programming, pp. 611–619 (1998)
16. Eberhart, R.C., Simpson, P., Dobbins, R.: Chapter 6, pp. 212–226. AP Professional, San Diego, CA (1996)
17. Fogel, D.B., Fogel, L.J., Atmar, W., Fogel, G.B.: Hierarchic methods in evolutionary programming. In: Proceedings of the First Annual Conference on Evolutionary Programming, pp. 175–182 (1992)
18. Goldberg, D.E.: Genetic Algorithms for Search, Optimization, and Machine Learning. Reading, MA: Addison-Wesley (1989)
19. Hansen, N., Ostermeier, A.: Adapting arbitrary normal mutation distributions in evolution strageties: The covariance matrix adaptation. In: Proceedings of the IEEE International Conference on Evolutionary Computation, pp. 312–317 (1996)
20. Hansen, N., Ostermeier, A.: Completely derandomized self-adaptation in evolution strategies. Evolutionary Computation Journal **9**(2), 159–195 (2000)
21. Higashi, N., Iba, H.: Particle swarm optimization with gaussian mutation. In: Proceedings of the IEEE swarm intelligence sysmposium 2003, pp. 72–79 (2003)
22. Holland, J.H.: Concerning efficient adaptive systems. In: M.C. Yovits, G.T. Jacobi, G.B. Goldstein (eds.) Self-Organizing Systems, pp. 215–230. Spartan Press (1962)
23. Holland, J.H.: Adaptation in Natural and Artificial Systems. Ann Arbor, MI: MIT Press (1975)
24. Juang, C.F.: A hybrid of genetic algorithm and particle swarm optimization for recurrent network design. IEEE Trans. Syst Man Cybern - Part B: Cybern **34(2)**, 997–1006 (2004)
25. Kennedy, J.: The particle swarm: Social adaptation of knowledge. In: Evolutionary Computation, 1997., IEEE International Conference on, pp. 303–308 (1997)
26. Kennedy, J.: Bare bones particle swarm. In: Proceedings of IEEE International Conference on Evolutionary Computation, pp. 80–87 (2003)
27. Kennedy, J., Eberhart, R.: Particle swarm optimization. In: Proceedings of Conference on Evolutionary Computation (CEC), pp. 1942–1948 (1995)
28. Laarhoven, P.J.M., Aarts, E.H.L.: Simulated annealing: Theory and applications. Springer (1987)
29. Lovbjerg, M., Rasmussen, T.K., Krink, T.: Hybrid particle swarm optimizer with breeding and subpopulations. In: Proceedings of GECCO, pp. 469–476 (2001)

30. Luus, R., Jaakola, T.H.I.: Optmization by direct search and systematic reduction of the size of search region. AIChE Journal **19**, 760–766 (1973)
31. Mendes, R., Kennedy, J., Neves, J.: The fully informed particle swarm: Simple, maybe better. IEEE Transactions on Evolutionary Computation **8(3)**, 204–210 (2004)
32. Ozcan, E., Mohan, C.K.: Particle swarm optimization: Surfing the waves. In: Proceedings of the Congress on Evolutionary Computation, pp. 6–9. IEEE Press (1999)
33. Padhye, N.: PSO source codes. http://web.mit.edu/npdhye/www/Source-codes.html
34. Padhye, N., Bhardawaj, P., Deb, K.: Improving differential evolution through a unified approach. Journal of Global Optimization (in press)
35. Padhye, N., Branke, J., Mostaghim, S.: Empirical comparison of mopso methods: Guide selection and diversity. In: Proceedings of CEC, pp. 2516–2523 (2009)
36. Padhye, N., Deb, K., Mittal, P.: Boundary handling approaches in particle swarm optimization (2012)
37. Padhye, N., Mohan, C.K., Mehrotra, K.G., Varshney, P.: Sensor selection strategies for networks monitoring toxic chemical release. In: Poceedings of Sensor Networks Applications (SNA) (2009)
38. Pant, M., Thangaraj, R., Abraham, A.: A New PSO Algorithm with Crossover Operator for Global Optimization Problems, pp. 215–222. Springer-Verlag, Berlin /Heidelberg (2007)
39. Parsopoulos, K.E., Vrahatis, M.N.: Particle Swarm Optimization and Intelligence: Advances and Applications. Information Science Publishing (2010)
40. Rao, S.S.: Genetic algorithmic approach for multiobjective optimization of structures. In: Proceedings of the ASME Annual Winter Meeting on Structures and Controls Optimization, vol. 38, pp. 29–38 (1993)
41. Ratnaweera, A., Halgamuge, S.K., Watson, H.C.: Self-organizing hierarchical particle swarm optimizer with time-varying acceleration coefficients. Evolutionary Computation, IEEE Transactions on **8**(3), 240–255 (2004)
42. Reklaitis, G.V., Ravindran, A., Ragsdell, K.M.: Engineering Optimization Methods and Applications. New York : Wiley (1983)
43. Reyes-Sierra, M., Coello, C.A.C.: Multi-objective particle swarm optimizers: A survey of the state-of-the art. International Journal of Computational Intelligence Research **2**(3), 287–308 (2006)
44. Rudolph, G.: Convergence of evolutionary algorithms in general search spaces. In: Proceedings of the Third IEEE Conference on Evolutionary Computation, pp. 50–54 (1996)
45. Schwefel, H.P.: Evolution and Optimum Seeking. New York: Wiley (1995)
46. Shi, Y., Eberhart, R.: A modified particle swarm optimization. In: Proceedings of the IEEE international conference on evolutionary computation, pp. 69–73 (1998)
47. Shi, Y., Eberhart, R.: Parameter selection in particle swarm optimization. In: Proceedings of the 7th International Conference on Evolutionary Programming VII, vol. 1447, pp. 591–600 (1998)
48. Storn, R., Price, K.: Differential evolution – A fast and efficient heuristic for global optimization over continuous spaces. Journal of Global Optimization **11**, 341–359 (1997)
49. Tawdross, P., Koenig, A.: Local parameters particle swarm optimization. In: Hybrid Intelligent Systems, 2006. HIS '06. Sixth International Conference on (2006)
50. Zhang, W., Xie, X.: DEPSO: Hyrbid particle swarm with differential evolution operator. In: IEEE international conference on systems, man and cybernetics (SMCC), vol. 3410, pp. 3816–3821 (2003)