Improving a Particle Swarm Optimization Algorithm Using an Evolutionary Algorithm Framework

Kalyanmoy Deb and Nikhil Padhye
Kanpur Genetic Algorithms Laboratory
Department of Mechanical Engineering
Indian Institute of Technology Kanpur
PIN 208 016, India
Email: deb@iitk.ac.in, npdhye@gmail.com

KanGAL Report Number 2010003

February 21, 2010

Abstract

Particle swarm optimization (PSO) algorithms are now being practiced for more than a decade and have been extended to solve various types of optimization problems. Despite their wide-spread use, in this paper, we investigate standard PSO algorithms for their ability to (i) approach the optimal region and (ii) simultaneously focus in order to find the optimum precisely, in simplistic yet scalable optimization problems exhibiting unimodality. In this study, we observe that a number of commonly-used PSO settings are not able to handle both aspects of optimization as efficiently like that of a previously-known genetic algorithm designed to solve the class of unimodal problems. Here, we highlight the importance of linking different optimization algorithms for their algorithmic equivalence, such that if desired, ideas and operators from one algorithm can be borrowed to enhance the performance of another. We illustrate this aspect by first suggesting an evolutionary algorithm (EA) which is fundamentally equivalent to a PSO and then borrow and introduce different EA-specific operators to the constriction-based PSO. Contrary to the standard PSO algorithms, our final PSO, which performs similar to the existing GA (and outperforms the GA in some occasions), is developed by replacing PSO's standard child-creation rule with a parent-centric recombination operator. The modified PSO algorithm is also scalable and is found to solve as large as 500-variable problems in a polynomial computational complexity. We emphasize here that EA and related optimization researchers should put more efforts in establishing equivalence between different existing optimization algorithms of interests (such as various genetic and evolutionary and other bio-inspired algorithms) to enhance an algorithm's performance and also to better understand the scope of operators of different algorithms.

1 Introduction

In 1995, Kennedy and Eberhart (1995) introduced a new paradigm in bio-inspired meta-heuristics – Particle Swarm Optimization (PSO) – 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 genetic algorithms and evolutionary programming based on the structural similarities. Over the past decade, the PSO paradigm has gained a wide-spread popularity in the research community 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 on improving the optimizer's performance in various contexts in order to place them at par with evolutionary methods.

The original paper by (Kennedy and Eberhart, 1995) recognized that PSO's child update procedure is similar to a genetic algorithm (GA)'s crossover operator. Eberhart and Shi (1998) discussed a resemblance of PSO with an evolutionary algorithm and considered PSO as an evolutionary algorithm (EA). Importantly they recognized that each operator of a GA 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

present in a PSO. Further, a PSO does not have an explicit selection operator, but its use of global best and personal best solutions in child-creation acts as an inherent selection operator. Several other studies looked at the similarities and differences between GA and PSO, for example, (Angeline, 1998a). This study characterized a GA procedure (an evolutionary algorithm) as primarily a competitive evolutionary process, whereas PSO as cooperative one.

To us, one of the striking feature of a PSO is that its child creation operator uses an *individualistic* approach in a sense that each population member uses its past solution, current solution, its own personal best solution to create a new solution. The only information it borrows from the other population members is that of the globally best solution (if a fully-connected topology is assumed). In comparison, an EA allows its recombination operator to be applied among any number of population members and importantly relies its search on this recombinative aspect involving multiple population members for creating new and hopefully better solutions.

Although such qualitative similarities exist between PSO and an EA, most PSO applications still use the same standard particle update rule (we call here the child-creation rule) and adjust associated parameters to solve different problems. Despite the original developer's early realizations of its similarity with other evolutionary algorithms, not many studies exist which build algorithmic connections between PSO and any EA. If a realizable connection is build, when stuck, PSO researchers can borrow knowledge from EAs through an algorithmic connection to enhance the performance of the PSO algorithm. The opposite is also possible and can be exploited by EA researchers as well.

In this paper, we illustrate the advantage of building an algorithmic connection between a PSO and an EA in the context of solving a particular class of problems — unimodal problems or problems having a few optima. These are simple test problems, but they can be scaled to any number of variables, thereby allowing us to evaluate an algorithm's fundamental working principles. Due to the unimodal nature of these problems, an efficient algorithm, starting from away from the optimal region, must have an adequate diversity-preserving and convergence properties so that it does not result in a stagnation anywhere and maintains sufficient exploration capability to drive its search all the way to the optimum. A previous real-parameter genetic algorithm (GA), developed to better handle unimodal problems (Deb et al., 2002), reported competitive number of function evaluations in solving three test problems based on a systematic comparison with a number of evolutionary algorithms, including CMA-ES (Hansen and Ostermeier, 2000), evolution strategies (Schwefel, 1995), differential evolution (Storn and Price, 1997), and a classical optimization method. In this study, we consider the same problems and make an attempt to evaluate existing PSO algorithms compared to the best-known results. Simulations reveal that some standard PSO implementations with commonly-used parameter settings fail to perform well and final solutions are nowhere near the previously-reported GA results.

When faced with such a predicament, the existing PSO framework does not provide a way-out to solve such simple, unconstrained, and unimodal problems competitively with respect to existing benchmark algorithms. Instead of choosing a more fancy option (such as multiple swarms, adaptive change in parameters etc.), we attempt to design an EA which is algorithmically identical to the basic PSO approach and then borrow well-known EA operators (which are usually not used in a PSO) in a manner to preserve PSO's individualistic traits.

To achieve this, an archive-based evolutionary algorithm which is fundamentally identical to a standard PSO procedure is proposed for the first time. This linkage between PSO and EAs opens up ways a plethora of EA operations that can be implemented in PSO to improve its performance. We demonstrate this fact clearly by considering each and every operator used in the existing real-parameter GA algorithm (G3-PCX (Deb et al., 2002)) and extend the canonical PSO algorithm with these operators one at a time. It is important to highlight here that without establishing the connection between PSO and EAs, such ideas are difficult to implement, simply because it becomes a difficult task to know where and how an operator from one algorithm can be incorporated in another algorithm efficiently. In solving the specific unimodal problems, we have borrowed ideas and operators, such as steady-state update of population, an explicit mutation operator, an explicit tournament based parent selection operator, a preselection operator from the earlier GA studies. We observe that there is a significant improvement in the performance of the modified PSO algorithm. Finally, we take a bold step in replacing the PSO's usual child-creation rule with the PCX recombination rule involving the same four solutions usually used in PSO (an individual's previous solution, current solution, its personal best solution, and globally best solution) so that PSO's individualistic trait is retained and the resulting algorithm can still be termed as a PSO algorithm. When this is done, the resulting PCX based PSO performs almost similar to the existing GAs. This appears

to be a remarkable achievement, particularly since standard PSO settings do not work up to the desired accuracy, and by simple analogy to an equivalent EA we were able to introduce key elements to the PSO to solve the problems with a respectable performance.

Our study here should not be misunderstood to say that if there already exists an EA to solve the problems of our interest, why do we care solving them by another optimization paradigm? Our aim here is not also to state that since PSOs and EAs have similar working principles, we should abandon one of these two methods. We realize that every algorithm or search paradigm has its own advantages and has its own niche of problems where it becomes a natural or a more effective choice than others. The motivation of this paper is to clearly emphasize the importance of establishing an algorithmic equivalence among different optimization algorithms, so that, if needed, ideas from one can be borrowed to enhance the performance of the other. As observed in this paper, while the ideas from EAs helped to improve the performance of canonical PSO, on one of the test problems the performance of enhanced PSO has surpassed that of the best-known EAs. We argue that the procedure adopted in this study is generic and can be suitably executed to develop efficient collaborative algorithms for solving other optimization problems. In addition, such tasks will provide researchers with valuable insights and understanding of positive and negative aspects of each algorithm and their scope in handling different complexities so as to facilitate the design of better and efficient algorithms.

In the remainder of this paper, we begin by providing a brief discussion on salient PSO studies. Thereafter, we present an evolutionary optimization framework (EO) which is essentially identical to PSO 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 GA study. PSOs with standard parameter settings are then applied to these problems. 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. After the final algorithm is developed, a detailed parametric study is made to fine-tune the PSO algorithm. The final PSO algorithm is applied to the same problems with larger number of variables, up to 500 variables, and is found to show a polynomial computational complexity. Finally, conclusions are made and importance of this study is highlighted.

2 Related PSO Studies

Popular studies in PSO have focused on the task of single objective optimization. However, the swarm intelligence idea has been successfully applied to handle other optimization tasks as well, like multi-modal optimization (Barrera and Coello, 2009), multi-objective optimization (Padhye et al., 2009a; Coello and Lechuga, 2002), including applications to real-world problem solving. Few attempts in the past have also been directed towards the development of theoretical models for particle trajectories in PSO. These proposed models have been validated empirically and often been found useful in explaining the swarm behavior, for example, see (Clerc and Kennedy, 2002; Kennedy, 2003; Ozcan et al., 1999). A detailed and complete recollection of past works is beyond the interest of this paper. For an extensive overview of past works in PSO, readers are referred to a recent survey (Banks et al., 2007). For reviews on multi-objective optimization, multi-modal optimization, dynamic optimization, readers are referred to (Reyes-Sierra and Coello, 2006; Barrera and Coello, 2009; Blackwell and Branke, 2006). Next, we revise some well-known extensions of PSO which have shown significant improvement over the canonical form.

The canonical form of PSO consists of a population of particles, known as a swarm, with each member of the swarm being associated with position vector $\mathbf{x}^t$ and 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 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' indicates the best position attained by a particle so far, whereas 'gbest' indicates the best location found so far in the entire swarm (this study implements popularly used fully informed swarm topology, i.e. 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 iteration t:

$$\mathbf{v}_i^{t+1} = w\mathbf{v}_i^t + c_1 r_1(\mathbf{p}_{l,i}^t - \mathbf{x}_i^t) + c_2 r_2(\mathbf{p}_g^t - \mathbf{x}_i^t), \quad (1)$$

$$\mathbf{x}_i^{t+1} = \mathbf{x}_i^t + \mathbf{v}_i^{t+1}. \quad (2)$$

Here, r_1 and r_2 are random numbers in $[0, 1]$, and w, c_1 and c_2 are pre-specified constants. In each iteration, every particle is updated serially according to 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 above stated child-creation is the position of the globally best solution $\mathbf{p}_g$, due to which the population aspect of the PSO is justified. In this study, we treat this individualistic aspect as a trademark feature of a PSO algorithm and argue that this is one of major difference of child-creation rule between a PSO and an evolutionary algorithm.

The parametric study on coefficients (w, c_2, 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 (Clerc, 2006; Shi and Eberhart, 1998b) 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 (Eberhart et al., 1996). The clamping mechanism restricted the velocity component in $[\text{-}v_{\max,j}\ v_{\max,j}]$ 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 $\mathbf{x}$ along the dimension. However, the clamping does not necessarily ensure that particles remain in the search space. It is worth noting that in unbounded search spaces such a 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'. In initial phases of the search, wide explorations are favored, whereas towards the end more focused search is desired. To achieve this, the concept of decreasing inertia weight (w) was introduced in (Shi and Eberhart, 1998a). 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 (Ratnaweera et al., 2004; Tawdross and Koenig, 2006).

Another major issue in PSO has been its pre-mature convergence. The particles often accelerate towards the swarm's best location and the population collapses. Keeping the particles within the search is still unresolved. Bringing back the particles into the search space, based on some strategies, drastically affects the PSO's performance. In the pursuit of solutions to these concerns, classified under swarm stability and explosion, 'constriction factor' (χ) was proposed in (Clerc and Kennedy, 2002). In the presence of χ, the velocity update equation 1 becomes:

$$\mathbf{v}_i^{t+1} = \chi \left(\mathbf{v}_i^t + c_1 r_1 (\mathbf{p}_{l,i}^t - \mathbf{x}_i^t) + c_2 r_2 (\mathbf{p}_g^t - \mathbf{x}_i^t) \right). \tag{3}$$

Equation 3 is currently one of the most popular version of velocity update rule in PSO studies.

Another related topic of our 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. Some representative works done in this direction are briefly mentioned as follows: In (Angeline, 1998b), 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 (Kennedy, 1997) and then creating artificial niches. The approach was successful in showing better convergence. In (Lovbjerg et al., 2001), 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 (Zhang and Xie, 2003), 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 (Higashi and Iba, 2003), a Gaussian mutation was combined with velocity update rule to improve PSO's performance. In (Juang, 2004), 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 (Pant et al., 2007), 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 EAs 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?' To our knowledge, there has been no significant effort spent in PSO or EA literature in this direction.

In this paper, we attempt to understand their similarity and differences by presenting an equivalent EA for a standard PSO algorithm. Once such an equivalence is established, knowledge of one algorithm can be efficiently transferred into the other. In the following section, we draw one such equivalence and subsequently illustrate how such connection can be utilized to improve the performance of PSO algorithm.

3 Similarities Between PSO and EO Algorithms

Both PSO and EAs are population based optimization procedures. As already mentioned elsewhere (Kennedy and Eberhart, 1995), 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 an EA. We illustrate this aspect in this section, by first outlining the essentials of an evolutionary optimization (EO) procedure.

An evolutionary optimization algorithm has a few generic properties. First, an EO works with a population of solutions in each iteration. Second, a selection operator chooses a few good solutions from the population based on objective and constraint functions. Third, present and/or past good solutions (parents) are utilized to create new (children) solutions by using essentially two types of operations: recombination and mutation. Although these terms are borrowed from natural genetics, there is a distinct purpose for each of them in an EO. Keeping their meaning from nature, the recombination and mutation operations can be understood as follows:

Recombination operator: Each recombination event involves *more than one* evolving members (parents) in creating a new solution (child). An evolving solution is one that is not fixed throughout a run, but is updated directly or indirectly 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. In this sense, if an update on a current population member involves a fixed variable vector (constant throughout the EO run, say the best population member of the initial population), it is 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 method, it is still a mutation operation, despite the 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 EO populations.

Of course, there can be other arguments or a disagreement with the above way of differentiating recombination and mutation operators for an EO algorithm, but for a proper understanding of their roles in an EO, a clear definition is helpful. The above understanding, motivated from nature, makes the properties and purposes of both operators clear and we use them here for our arguments.

After the creation of a new population (or a subpopulation) through selection, recombination and mutation operations, it is usually compared with current (and previous) population, and the better solutions (of size equal to EO population) are declared as the next generation population. This procedure is known as an elite-preserving operation. One or more termination criteria are then checked to decide on the termination of the EO procedure. To keep a record and to utilize previous population-best solutions in subsequent generations, often an external population (commonly known as an *archive*) is maintained and updated at the end of each generation. The optimized solution is the best solution stored in the final archive.

Based on the above general working principle, let us now consider a specific EO algorithm, as follows. In addition to the usual population P^t of size N at generation t, we also consider an archive population A^t of the same size N. To start with, the archive population is identical to the evolutionary 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 ($\mathbf{p}_g^t$). For the i-th evolutionary population member $\mathbf{x}_i^t = P_i^t$, we use three (parent) solutions – (i) the population member, $\mathbf{x}_i^t$, (ii) its corresponding archive partner, $\mathbf{p}_{l,i}^t$, and

(iii) the globally best archive member, $\mathbf{p}_g^t$ – to create a new (child) solution using the following equation 4.

$$\mathbf{x}_i^{t+1} = \mathbf{x}_i^t + c_1 r_1 \left(\mathbf{p}_{l,i}^t - \mathbf{x}_i^t\right) + c_2 r_2 \left(\mathbf{p}_g^t - \mathbf{x}_i^t\right). \quad (4)$$

This 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 sketches the procedure for one population member and its archive partner. All evolutionary population members are used with their corresponding archive partner and $\mathbf{p}_g^t$

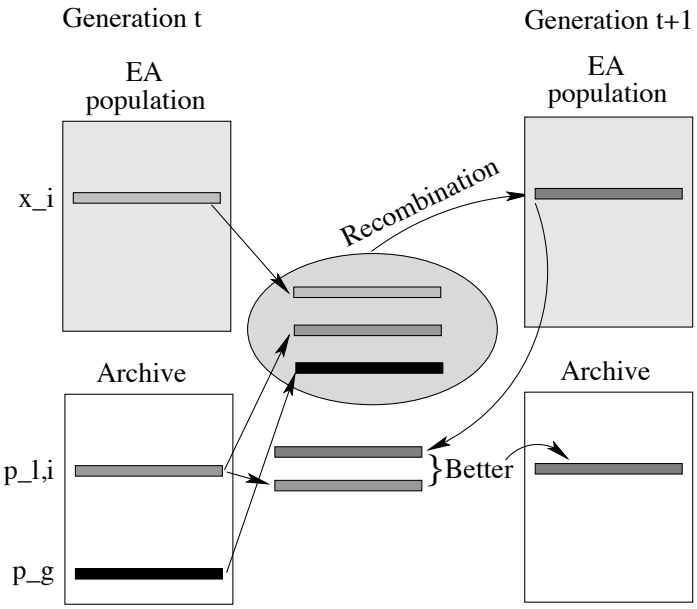

Figure 1: A sketch of an EO having a recombination operator involving three evolving solutions.

in the above child-creation process and the corresponding archive members are updated. When all N evolutionary population members are created and archive members are updated, the globally best archive solution for the next generation ($\mathbf{p}_g^{t+1}$) is declared as the best archive population member. This procedure continues till a termination criterion is met.

The above procedure is a generational approach 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 mutation operation used in this EO algorithm. The archive population gets updated along with the evolutionary population, but no explicit evolutionary operations are performed independently on the archive population.

A little thought will reveal that the procedure described above is also a standard PSO algorithm without the velocity term. We suggest a modification to the above evolutionary procedure a little later to design a similar velocity-based PSO algorithm. But the similarity of PSO without the velocity term and the suggested evolutionary optimization algorithm is striking. In fact, both algorithms, although look different at the outset, are identical algorithms. The EO algorithm uses a distinctive recombination operator. Due to the algorithmic similarity established herein, the PSO algorithm is also engaged in a recombination operation on its evolving population members.

In an usual EO algorithm, the creation of the new population solely depends on the current population members (and if an archive is used, the current archive members). In this sense, an usual EO algorithm is a Markov chain. But an EO need not be always a Markov chain and one can design an EO algorithm which uses previous populations as well in creating the new population. In this spirit, we modify the previously presented EO algorithm by including i-th population member from the $(t-1)$-th generation in the recombination operation, as follows:

$$\mathbf{x}_i^{t+1} = \mathbf{x}_i^t + w\left(\mathbf{x}_i^t - \mathbf{x}_i^{t-1}\right) + c_1 r_1 \left(\mathbf{p}_{l,i}^t - \mathbf{x}_i^t\right) + c_2 r_2 \left(\mathbf{p}_g^t - \mathbf{x}_i^t\right). \quad (5)$$

Figure 2 shows a sketch of the procedure in which four solutions are involved in the recombination process. The above recombination operator uses four evolving solutions to create a single child solution. As

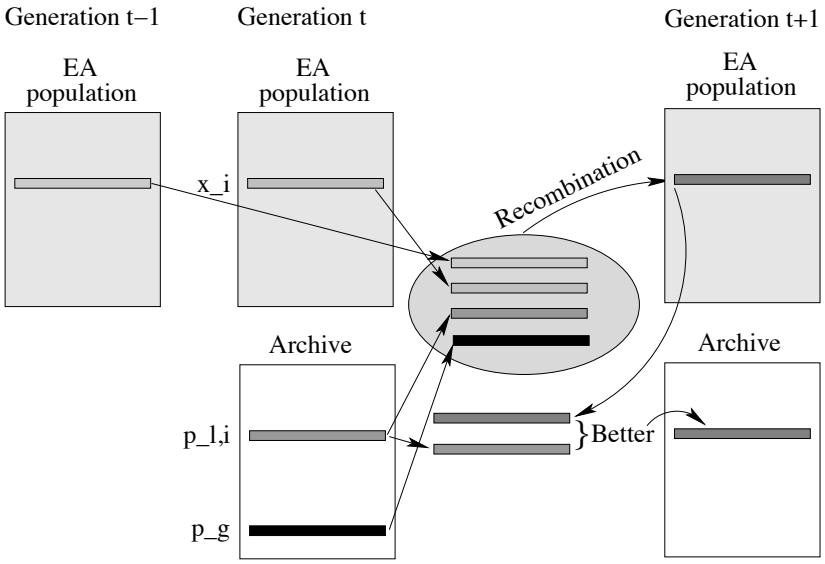

Figure 2: A sketch of an EO having a recombination operator involving four evolving solutions.

mentioned above, this is an uncommon recombination operation (due to the use of $\mathbf{x}_i^{t-1}$), but definitely qualifies as a recombination operation due to the use multiple existing evolving solutions as parents.

Often in PSO studies, an additional mutation operation is performed on $\mathbf{x}_i^{t+1}$ (Higashi and Iba, 2003). Such an operation is similar to the standard mutation operation usually performed usually after the recombination operation in an EO algorithm.

The above argument clearly shows that for the standard PSO algorithm, we can design an equivalent EO algorithm which is algorithmically identical to the corresponding PSO algorithm. We now discuss an advantage of considering the above simile with an EA.

3.1 Advantages of Seeking Similarities Between 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 practices in this direction are to perform a parametric study with its parameters, such as w, c_1, c_2. Some latest studies include the use of non-uniform probability distributions for r_1 and r_2 as well (Clerc, 2006; Kennedy, 2003) or the use of multiple swarms etc. (Mendes et al., 2004). But most of the modifications are based around the child-creation rule given in equation 5. If the update rule is to be strictly followed in all PSO studies, it may possess strict restrictions to the wide-spread use of PSO in practice and may give a myopic view of the PSO's potential as an optimization algorithm. A clear understanding of each term in the PSO child-creation rule and their equivalence with a similar EO algorithm may open up ways by which children solutions can be created in different scenarios. Embracing the equivalence with an EO procedure will allow us to use plethora of other EA strategies and operations, that have been developed since the inception of genetic algorithms, evolution strategies and evolutionary programming since many decades, to the PSO framework almost in a straightforward manner.

The discussions in the previous subsection indicate that a major way an EO algorithm differs from a PSO algorithm is that the latter has a history term (the velocity term $w(\mathbf{x}_i^t - \mathbf{x}_i^{t-1})$), which is missing in most EO algorithms. As PSO can benefit from EO's vast literature, EO algorithms can also benefit the same way from PSO's distinctive operations, such as the use of the velocity or the history term. The similarity study made in the previous subsection allows us to determine fundamental differences between the two algorithms and provides us with ideas which can be borrowed from one algorithm to enhance the performance of the other.

In the next section, we exploit the similarities and concentrate on solving a specific type of optimization problems. We demonstrate how some well-known evolutionary operations can be included in a PSO

framework to enhance its performance that are otherwise difficult to comprehend from the usual PSO framework.

4 PSO for Optimizing Unimodal Problems

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 to progress towards the optimal region and then to focus to find the optimum with a specified precision. A previous study considered a number of evolutionary algorithms like, generalized generation gap (G3) model using a parent-centric crossover (PCX) operator, differential evolution, evolution strategies (ESs), CMA-ES, and a classical method on following test problems (Deb et al., 2002):

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

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

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

In all these problems, we use $n = 20$ and later present results for n as high as 500. 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 restricting $x_i \in [-10, -5]$ for all i, in all problems, but in subsequent generations we do not confine solutions to lie in the above range. Most PSO studies, while solving numerical test problems, initialize their population randomly around the optimal solution (since the optimal solution is known a priori for a test problem) and use different strategies to force the solutions to lie within the specified search space 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. Initializing population around the known optimum does not also allow to test an algorithm's ability to approach towards the optimal region, rather in most cases it tests the algorithm's ability to focus close to the optimal solution. This is how most classical optimization algorithms are designed. An efficient algorithm, starting from an initial guess solution must first approach the optimal region and then focus in the region to find the optimal solution with a desired precision. In this study, we investigate both aspects of an algorithm by considering two evaluation criteria.

First, after initializing the population randomly away from the optimal region, we count the number of function evaluations needed for the algorithm to find a solution close to the optimal solution and we call this our first evaluation criterion S_1. We arbitrarily choose a limiting function value of 0.1 for this purpose. 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 function evaluations provides us with an idea of the algorithm's ability to focus and also to find a solution with a precision. An optimization algorithm must perform well in both the aspects. In particular, the second criterion (S_2 for which $F \leq 10^{-20}$) provides an overall performance measure of an optimization algorithm for achieving both aspects needed in a generic optimization task.

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

Deb et al. (2002) proposed a steady-state genetic algorithm which solved the earlier described test problems in a computationally fastest 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).

To summarize, the G3-PCX algorithm has following 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 for at most N function evaluations, they can become parent as quickly as after only two function evaluations. For solving unimodal problems, this property can provide an adequate selection pressure for good solutions.
- It uses an elite-preserving operator. Since children are compared against parents before accepting them to the population, elites of population that are not bettered by offspring are kept in the population.
- It involves the globally best solution in every child-creation process. For solving unimodal problems in a computationally fast manner, this operation helps in progressing the current best solution towards the optimum solution.
- It uses a recombination operator that creates child solutions around the globally best solution. Again, this helps in quickly approaching towards the optimum solution for unimodal problems.

The earlier extensive study on G3-PCX algorithm reported the best, median and worst number of function evaluations needed based on 50 different runs on the three problems with the S_2 criterion. Table 1 presents those results. In a comparison with a CMA-ES (Hansen and Ostermeier, 1996, 2000), a couple

Table 1: G3-PCX results, as reported in (Deb et al., 2002).

	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

of self-adaptive evolution strategies (Schwefel, 1995; Beyer, 1995), differential evolution (Storn and Price, 1997) and a quasi-Newton classical method (Reklaitis et al., 1983), the function evaluations reported in the above table were found to be the smallest. The F_{ros} problem has another local optimal solution and G3-PCX is able to converge to the globally optimal solution in 38 out of 50 runs. Thus, we can assume that the G3-PCX algorithm is a computationally efficient algorithm for solving all the three problems used in this study.

4.2 Standard PSO Algorithms

Several PSO settings are applied on the 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, each algorithm is tested on a test problem 50 times (each run starting with a different initial population). A particular run is concluded if the evaluation criterion (S_1 or S_2 depending on which one is chosen) is met, or the number of function evaluations exceed one million. If only a few out of 50 runs are successful then we report the count of successful runs in the bracket. 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 with a maximum of one million function evaluations, we denote this by marking with 'DNC' (Did Not Converge). In such 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.

Here, 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 (Padhye et al., 2009b), 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 (Clerc, 2006) 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 (Clerc, 2006). Linearly decreasing w promotes a wider search in initial phases and focused search towards the end.
4. Equation 3 with ($\chi = 0.729$, $c_1 = 2.05$, $c_2 = 2.05$): As discussed earlier this modified PSO move prevents swarm explosion and ensures stability. The parameter settings have been taken from (Clerc and Kennedy, 2002).

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 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. For solving unimodal problems, the performance of the above PSO is strikingly poor, compared to the G3-PCX algorithm. 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 larger than zero. These results indicate that when PSO is set to evolve freely without its search restricted within any variable bounds, a standard setting of its parameters makes its search wander in the search space and in the process is not able to come close to the true optimum after a considerable number of evaluations.

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* (DNC)	*2,577.5* (DNC)	*3,709.4* (DNC)	*3073.2* (DNC)	*6,279.9* (DNC)	*12,756.5* (DNC)	*435,400.1* (DNC)	*896,497.6* (DNC)	*1,419,791.0* (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* (DNC)	*3.99* (DNC)	*11.53* (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* (DNC)	*4.84e-04* (DNC)	*3.99* (DNC)

Next, another experimentation is conducted based on a more popular parameter settings 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 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 S_2 criterion values for both these problems are much worse than those obtained using the G3-PCX algorithm (Table 1). For the problem F_{ros}, 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 $1.50(10^{-11})$. Higher 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, making a remarkable convergence pattern.

Since the above parameter setting performed well, we try a modification to the same setting by linearly 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 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 $F_{\rm ros}$ are 39 out of 50. Since the PSO works with an unbounded search space the χ factor, it is useful in preventing disproportionate increase in velocity. Due to an overall good performance in general on all three problems, in rest of this paper, we shall use the PSO with the constriction factor based update rule.

The above four parametric 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. Overall, the performance of the standard PSOs on the chosen test problems is quite inferior when compared to those found using the G3-PCX algorithm (Table 1).

Under the circumstances where PSOs with well-known parameter settings for the child-creation rule and population sizes are unable to find accurate results in repeated applications from different initial populations with a reasonable number of function evaluations, what options do PSO practitioners have to progress further? The PSO literature offers a number of other complicated options: (a) Instead of using static values of c_1 and c_2 throughout a run, these parameters 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 (Clerc, 2006). 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 such complex nature of these concepts and additional requirement of setting parameters, the dynamics of the evolution of good solutions in a problem may not be obvious and if the algorithms fail, there is no straight-forward way to find clues in order to set the parameters appropriately. Particularly, for simple problems such as finding the minimum of unimodal problems, an efficient optimization procedure need not rely on complicated operations and need not be sensitive to a number of parameters. Rather, a simple yet effective strategy must be adopted.

5 Modifying PSO with Ideas Borrowed from Evolutionary Algorithms

In this section, we borrow several ideas from the evolutionary algorithm literature and use our PSO-EO connections discussed in Section 3 to try to enhance PSO's performance as above.

5.1 Using a Mutation Operator

First and foremost, we realize that a mutation operator can be applied immediately 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 (Higashi and Iba, 2003). The concept resembling the mutation in PSO was originally introduced as 'craziness'. This has became 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, thereby avoiding a pre-mature convergence. *Random*, *Gaussian*, *Cauchy*, etc., distribution based mutations have been employed in past for this purpose (Clerc, 2006) and often reported to improve the performance of PSO. In majority of past studies in PSO a bounded search space is used in which defining the parameters or the range of any distribution is relatively simple. Since, we are working with an unbounded search space, the operating space for mutation is defined in the region between a particle and '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. The introduced 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 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

```
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=gbest.x_i - δ*(particle.x_i-gbest.x_i);
        High=gbest.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
```

Figure 3: Proposed mutation operator for PSO.

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 in 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)

performs better on F_{elp} and F_{sch} than that 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 targetted accuracy of the order 10^{-20}, but 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 the 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 $t_f = 0.25$ – with the G3-PCX algorithm (benchmark optimizer for unimodal problems) based on a *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 after every few function evaluations for all three algorithms and compare them to investigate if they behave in a similar or dissimilar manner.

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 three algorithms. We plot these figures till a function value of 0.1 is achieved. The figures show that the diversity

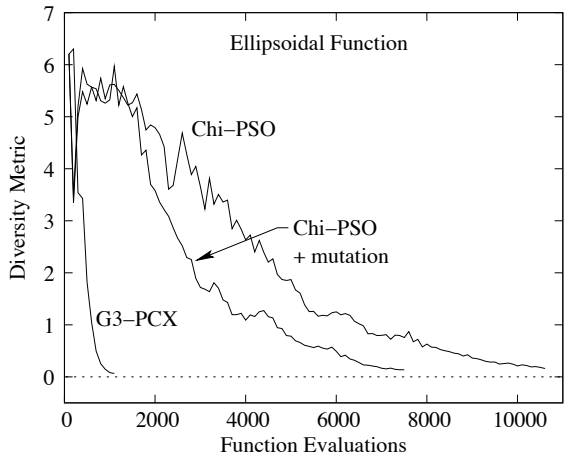

Figure 4: Diversity metric for $F_{\rm elp}$ function for χ-PSO with $t_f = 0.0$, $t_f = 0.25$, and G3-PCX.

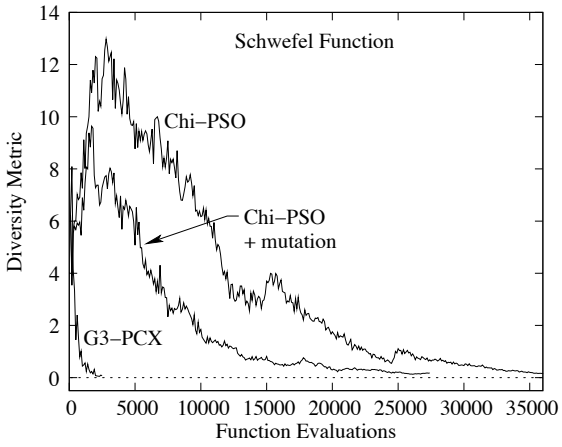

Figure 5: Diversity metric for $F_{\rm sch}$ function for χ-PSO with $t_f = 0.0$, $t_f = 0.25$, and G3-PCX.

of the top half of the population under G3-PCX algorithm shows some 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, its population diversity may also follow a similar trend.

We observe that the population diversity of original χ-PSO is quite different and slow compared to that of G3-PCX. However, when the mutation operator is added, the diversity metric variation gets closer to that of G3-PCX. Interestingly, all three algorithms have a similar behavior – diversity increases in the beginning and then reduces, except that their rates of rise and fall of diversity are different. Since the function evaluations reported in Table 3 with mutation are also better than those found without any mutation in Table 2, we conclude that by achieving the rise and fall of population diversity quicker is a better strategy in solving unimodal problems of this study. We have observed a similar outcome with $F_{\rm ros}$ and do not show the plot here.

With the success of introducing a mutation operator in χ-PSO, we now add a few other standard evolutionary algorithm operators one by one to hopefully reduce the difference in performances between the resulting PSO and G3-PCX. In all these efforts, we maintain the individuality aspect of PSO (discussed in Section 2) and work around modifying the child-creation rule.

5.2 Steady-State Update of Global Best Solution After Every Child-Creation

In a standard PSO algorithm, the same global best solution is used for all N population members to create N children solutions. If we think of the PSO procedure as an equivalent EO procedure as discussed in Section 3 and compare that EO 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, after every child-creation and its evaluation, it 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. In this *elite-preservation* process, if a good child solution is created it can immediately start contributing in the creation of new solutions by participating in recombination with other population members. In solving unimodal (or likewise) problems, such a steady-state algorithm is useful in general and was reported to be better in the G3-PCX study (Deb et al., 2002). This implementation, if done on the PSO algorithm, makes a small change in the existing PSO algorithm (namely, the global best solution is updated after every child-creation) and the resulting algorithm can still be considered as a PSO algorithm.

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 $F_{\rm elp}$ (in terms of median performance) and $F_{\rm sch}$. The algorithm is still not able to solve $F_{\rm ros}$ with desired accuracy.

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 in 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)

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 quickly.

Importantly, this simple idea may not have come to one's mind easily if one restricts only to the PSO child-creation rule. An equivalence of PSO with a similar evolutionary algorithm framework (G3-PCX here) and the importance of the steady-state operation there for solving similar problems allows us to modify the PSO algorithm to a steady-state PSO algorithm, which seems to make a difference in the performance of the PSO algorithm in solving unimodal test problems.

We have also tested the performance of purely mutation driven steady-state PSOs by switching off the velocity and position updates, and found poor 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 Preselection: Comparison of Parent and Child

The above modification was interesting enough for us to look for further updates by which we can enhance the performance of the PSO algorithm, motivated by the operators used in the G3-PCX algorithm. The G3-PCX algorithm replaced a randomly chosen solution from the population with the better of newly created (child) solutions and the chosen solution. Since in a PSO framework, every population member has a meaningful ancestor (corresponding population member from the previous generation), an update of a random population member will make its previous corresponding population member meaningless. However, the concept of a comparison of a population member with its newly created child and acceptance of the better solution can in general be a good strategy in maintaining diversity and is suggested as early as in 1970 in the context of *preselection* in the EA studies (Cavicchio, 1970).

5.3.1 Steady-State and Parent-Child Comparison

First, we implement parent-child comparison (preselection) approach along with the steady-state approach on the mutation-based PSO. Table 5 shows that the obtained results are not as good as the mutation-based approach (Table 3). The performances are also very poor compared to mutation-based with steady-state approach (Table 4). Due to an emphasis of newly created better solutions with the steady-state approach,

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 in 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)

the inclusion of preselection introduces additional selection pressure for the currently best population members. This causes a degradation in the performance of the overall procedure.

Thus, so far, the best performance is observed to come from 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

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 in 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)

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

5.4 Selection of Parents

Since the steady-state approach has made a positive change 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 more and in the presence of steady-state procedure, the performance of the overall algorithm may be better. In an evolutionary algorithm, better parents are usually selected for child generation by a separate *selection* operator. With this principle of EAs in mind, we introduce two selection operations one by one on the mutation-based steady-state PSO procedure discussed in subsection 5.2.

First, we use a random selection operation, in which a population member is chosen randomly for creating a new child solution, instead of starting from the top of the population and moving down the population one by one, as done in a standard PSO. 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 not using this operator at all. Table 7 presents the results. In a comparison with Table 4, this table shows that the inclusion of random selection does not

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 in 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)

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) operator to select a parent solution. The 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

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 in 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)

performances on other two problems are not better than the best results obtained so far,

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

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-

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 in 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)

child comparison, in general, is not a good approach with PSO and we do not continue with this operation any further.

5.6 Discussion on Results Obtained Using PSO Child-Creation Rule

From a standard PSO approach, we have shown how the concepts from evolutionary algorithm literature can be borrowed to enhance the performance of PSO in solving a particular class of problems (exhibiting unimodality). In this study so far, we have used the standard PSO child-creation rule with parameters obtained after a detailed study, discussed in Section 4.2. Several EA operators are then introduced through the PSO-EO 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 (Deb et al., 2002) 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 with our extensive simulations with various combination of operators is 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.

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)

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, etc.

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

6 Modifying PSO Further with Different Solution Update Rules

The PCX operator requires three or more parent solutions and uses a parent centric recombination operator in which the probability distribution is computed from the vector differences of the parents utilized in creation of child solutions (Deb et al., 2002). The probability distribution is centered around the currently best parent solution. Figures 6 and 7 compare the region from which a child is chosen using PSO child-creation rule (without the velocity term) and the PCX operator on three parent solutions on a two-variable problem. The PCX update rule is described below.

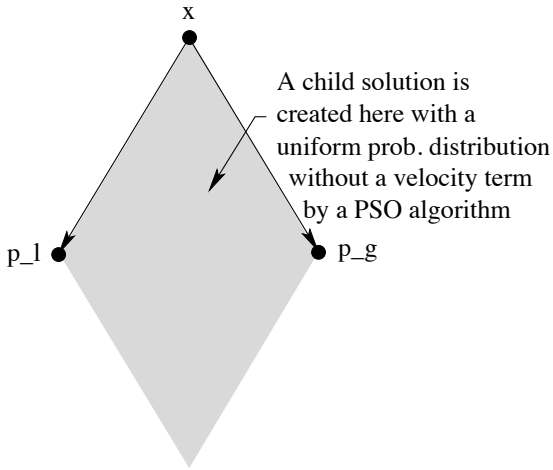

Figure 6: For three solutions $\mathbf{x}$, $\mathbf{p}_l$ and $\mathbf{p}_g$, the region for creating a child solution is shown using PSO algorithm without a velocity term.

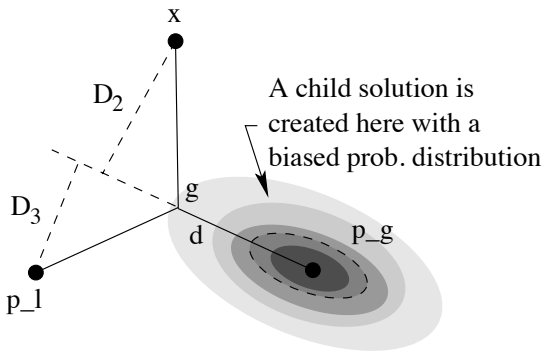

Figure 7: For three solutions $\mathbf{x}$, $\mathbf{p}_l$ and $\mathbf{p}_g$, the region for creating a child solution is shown using the proposed PCX update rule.

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:

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

where $\vec{e}^{(i)}$ are the $(\mu-1)$ orthonormal bases that span the subspace perpendicular to $\mathbf{d}^{(p)}$. 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-EO simile 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 EA literature. Here, we borrow the blending idea from PCX operator Deb et al. (2002) 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. However, this change in the child-creation process still maintains the PSO's individualistic trait, as discussed before. Our effort in drawing the simile allows us to incorporate such changes at a stage where we have exhausted with different PSO variants with various parameter settings and with an infusion of various evolutionary algorithm operators to the standard PSO algorithm. 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

Firstly, we do not consider the velocity term, that is, $w = 0$ (though there is no way to compute velocity here, we can still calculate velocity as $\mathbf{v}^t = \mathbf{x}^{t+1} - \mathbf{x}^t$). Thus, here the effect of completely replacing the PSO child-creation rule with the PCX update rule is investigated. Like in the PCX operator, child solutions are created around the globally best solution $\mathbf{p}_g$. Since we are working with a spatial distribution to create points and are not limited to the space spanned by the resultant of three vectors (as in the usual PSO update), there is sufficient exploration. From previous sections we adopt mutation and steady-state operations – two most effective operators, for the remaining simulations. 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 that presented in

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 in 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 10. The reason for such a performance can be as follows. In unimodal problems, the individual population members are expected to get improved in a continual manner and the personal-best solution $\mathbf{p}_l$ is most often likely to be identical to the population member ($\mathbf{x}$) itself. When this happens, the PCX operator produces a child solution along 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 7). 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, which we discuss 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

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 in S_2 so far	**5,600**	**6,200**	**6,600**	**29,000**	**33,900**	**40,600**	**61,100 (42)**	**80,600**	**93,400**

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. 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 near to 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 8, 9, and 10. 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. There is a drastic change in behavior of the PSO algorithm, which is evident

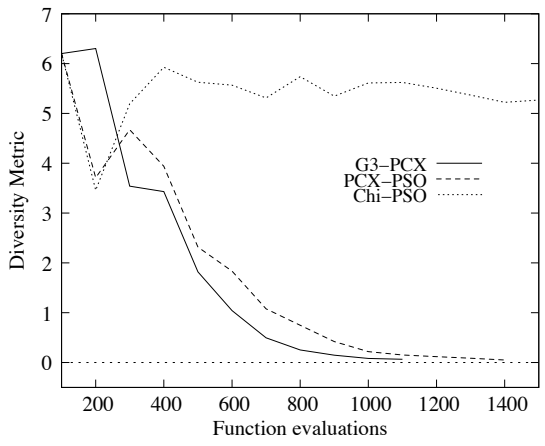

Figure 8: Diversity metric values for G3-PCX and PCX-PSO are similar for F_{elp}.

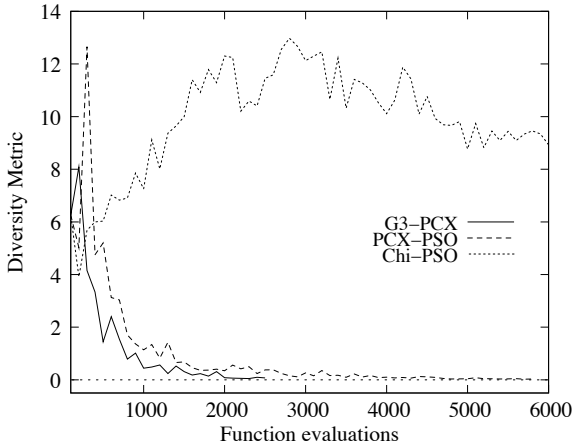

Figure 9: Diversity metric values for G3-PCX and PCX-PSO are similar for F_{sch}.

from the dynamics of the population diversity with generation. Importantly, if we did not draw the simile between the PSO algorithm and EA, it would have been difficult to think of such a modification.

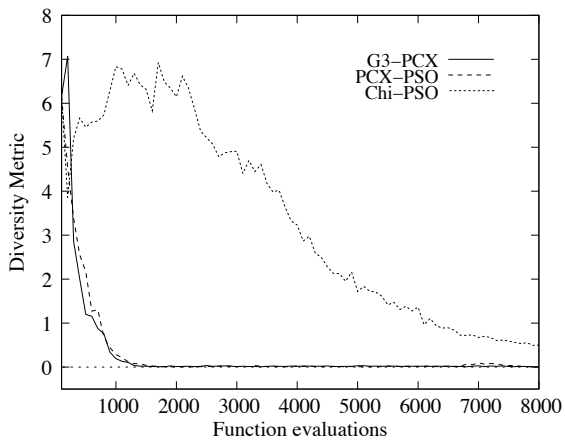

Figure 10: Diversity metric values for G3-PCX and PCX-PSO are similar for $F_{\rm ros}$.

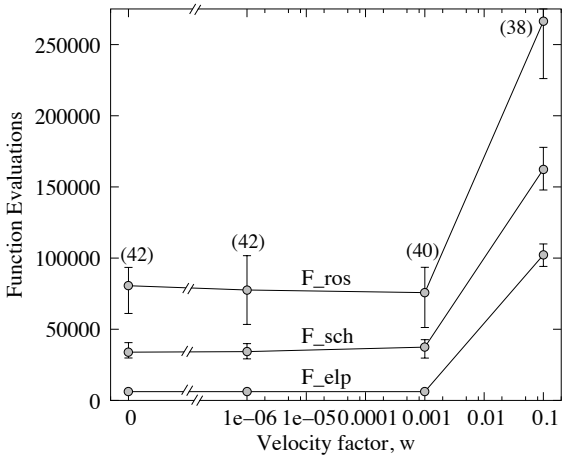

Figure 11: Parametric study with velocity factor w.

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. In this section, we now 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 11 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 is not greatly important for the proposed algorithm in solving the unimodal problems. However, a small w may improve the performance occasionally .

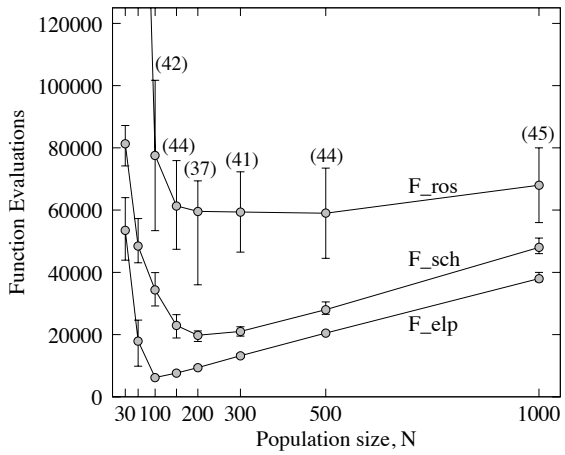

Figure 12: Parametric study with population size.

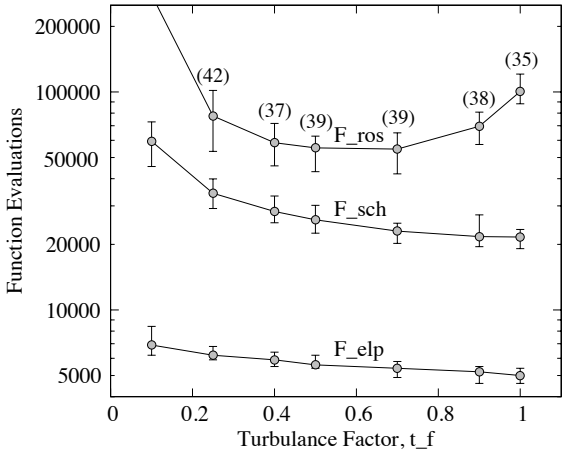

Figure 13: Parametric study with mutation factor t_f.

Next, we fix $w = 10^{-6}$ and $t_f = 0.25$ and vary the population size N. Figure 12 shows the obtained results. The problem $F_{\rm elp}$ is quite sensitive to the population size and works well for $N = 100$ and there-about. However, for $F_{\rm sch}$ and $F_{\rm ros}$ problems, our modified PSO performs well around $N = 150$ and $N = 500$, respectively. Too small population size does not allow 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 (Clerc and Kennedy, 2002).

Finally, we fix $w = 10^{-6}$ and $N = 100$ and vary the mutation probability t_f. Figure 13 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 it to work well.

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.

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

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 evolutionary algorithm to develop a methodology which performs comparable to the same evolutionary algorithm. Figure 14 summarizes the a 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.

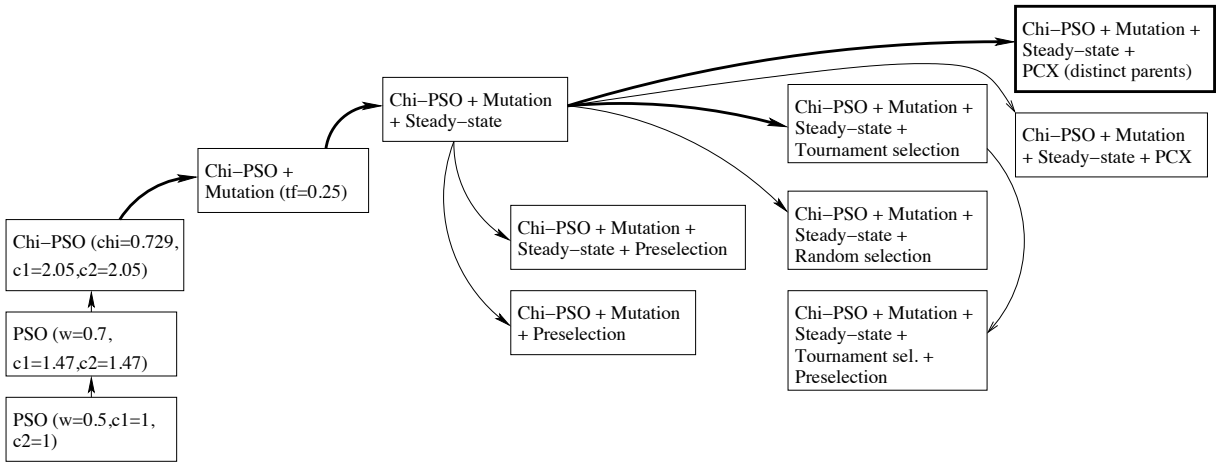

Figure 14: The flowchart of the PSO updates studied here.

The resulting procedure 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. However, the child-creation rule is different from the usual PSO update. It uses a biased distribution around the global best solution rather than a 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 evolutionary algorithms for solving similar problems in the past.

7 Scale-Up Study

Finally, motivated by the development of a successful PSO-based algorithm for 20-variable unimodal problems, we now test its performance in solving higher-dimensional 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 variables. For each test problem, the best, median and worst number of function evaluations in 10 runs required to meet a termination criterion of 10^{-10} are noted. As the dimension (n) of a problem is increased, we use a larger population size (N), a smaller mutation probability (t_f), and smaller values of w_η and w_ζ, as shown in Table 14. It is important to

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

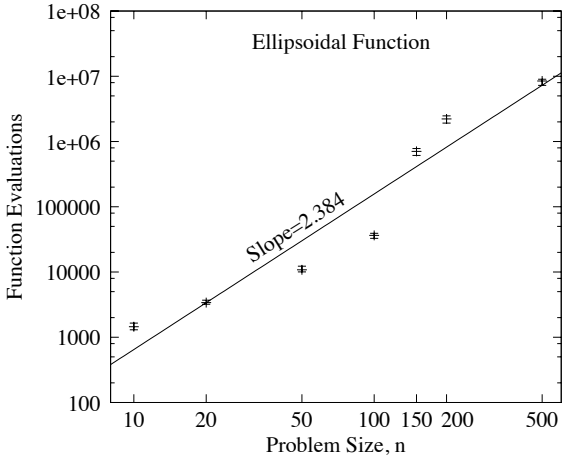

Figure 15: Scale-up study of the proposed PCX based PSO algorithm on F_{elp}. All 10 runs are successful.

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 depicted in Figures 15,

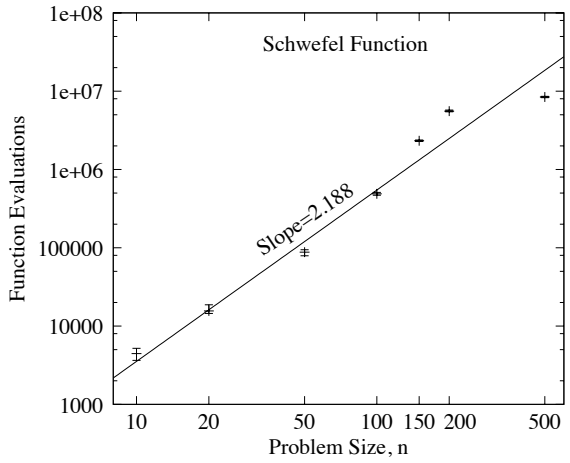

Figure 16: Scale-up study of the proposed PCX based PSO algorithm on F_{sch}. All 10 runs are successful.

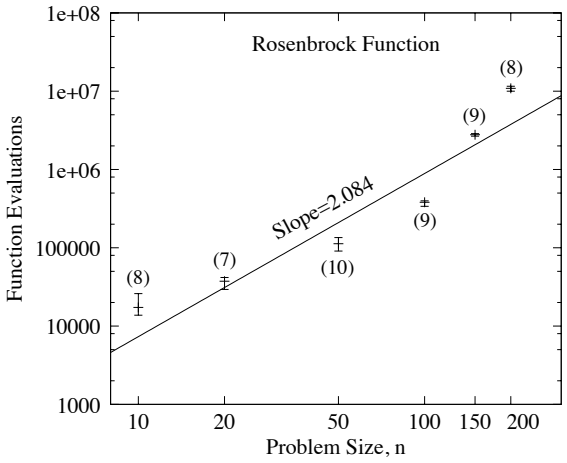

Figure 17: Scale-up study of the proposed PCX based PSO algorithm on F_{ros}. The numbers in brackets show the successful runs in 10 runs.

16 and 17. From the figures, it is evident that the number of function evaluations required by the PCX enabled PSO algorithm increases polynomially (with a degree between 2 to 3) with the dimensionality of the search space. The algorithm is able to converge in all 10 runs for F_{elp} and F_{sch}. For F_{ros} the number of

successful runs (out of 10 runs) are mentioned in the figure. These results amply demonstrate the proposed algorithm's ability to scale up to problems having a large number of variables. We are not aware of any previous PSO study in which problems having 200-500 variables are attempted to be solved.

8 Conclusions

Particle swarm optimization (PSO) algorithms are practiced since 1995, when its developers first proposed the idea of solving real-parameter optimization problems. Despite the use of a population in the search and an iterative evolution of its population members, its developers (at least, initially) did not want to 'mix-up' PSO with an evolutionary algorithm. The basic principle of a PSO lies in its famous child-creation rule and its individualistic approach which uses only a member's immediate past, present and best-ever to create a child solution. The only exception is the use of the globally best solution in its child-creation rule.

In this paper, we have drawn a simile of PSO with a recombinative archive-based evolutionary optimization algorithm which is algorithmically identical to a standard PSO algorithm. This is done not to demonstrate that a PSO is an EA (and possibly an EA is also a PSO), but to highlight the fact that such a similarity analysis allows us to borrow useful operations from one algorithm in enhancing the performance of another.

To demonstrate this, we have chosen three 20-variable unimodal test problems from the literature. We have found that standard PSOs with a number of commonly-used parameter settings are not able to solve the problems to the desired accuracy. Thereafter, we have modified the PSO operators by bringing in ideas from a previously-developed computationally-efficient genetic algorithm for solving the same problems based on the established simile between PSO and a specific EA. The use of an additional mutation operator, a steady-state approach, and a binary tournament for parent selection on a constriction based PSO has emerged to be the best strategy for solving two of the three chosen problems. Although the results are much better than a standard PSO, without all these additional operators, the performance of the modified PSOs were found to be an order of magnitude worse.

To look for a better modification of the PSO approach, we have finally replaced PSO's child-creation rule with a three-parent PCX operator. The parents are chosen in a manner so as to maintain the individualistic property of a PSO algorithm. The usual entities, such as the population member, its individual best solution, the globally best solution and member's immediate past solution are all used in the PCX update rule. In addition, the PSO is enhanced with a steady-state approach and an additional mutation operator is included. The resulting PCX-based PSO approach has shown to find a comparable performance to the G3-PCX algorithm. A sensitivity study of parameters associated with the PCX-based PSO approach has shown robustness of the approach. Finally, a scale-up study on as large as 500-variable problems has shown a polynomial computational complexity of the proposed approach.

To summarize the achievements of this paper, we highlight here that an algorithm can benefit from another algorithm by first understanding the similarities and dissimilarities 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. In this paper, we have shown this aspect between PSO and a specific EA in the context of solving unimodal problems for reducing computational cost for achieving a desired solution accuracy. This study should encourage more such collaborative algorithm development tasks for other optimization problems, such as multimodal optimization, multi-objective optimization, dynamic optimization, combinatorial optimization, just to name a few. Such studies should provide the algorithm developers useful insights to the fundamental working principles of different algorithms and help them develop better and more efficient collaborative algorithms.

References

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

Angeline, P. J. (1998b). Using selection to improve particle swarm optimization. In *Proceedings of IEEE Congress on Evolutionary Computation*.

Banks, A., Vincet, J., and Anyakoha, C. (2007). A review of particle swarm optimization. part i: background and development. *Natural Computing*, 6(4):467–484.

Barrera, J. and Coello, C. A. C. (2009). *A review of Particle Swarm Optimization Methods Used for Multimodal Optimization*, volume 248, pages 9–37. Springer.

Beyer, H.-G. (1995). Toward a theory of evolution strategies: Self-adaptation. *Evolutionary Computation Journal*, 3(3):311–347.

Blackwell, T. and Branke, J. (2006). Multiswarms, exclusion, and anti-convergence in dynamic environments. *IEEE Transactions on Evolutionary Computation*, 10(4):459–472.

Cavicchio, D. J. (1970). *Adaptive Search Using Simulated Evolution*. PhD thesis, Ann Arbor, MI: University of Michigan.

Clerc, M. (2006). *Particle Swarm Optimization*. ISTE Ltd, UK/USA.

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

Coello, C. A. C. and Lechuga, M. S. (2002). MOPSO: A proposal for mulriple objective particle swarm optimization. In *Congress on Evolutionary Computation, pages 825-830. IEEE*.

Deb, K., Anand, A., and Joshi, D. (2002). A computationally efficient evolutionary algorithm for real-parameter optimization. *Evolutionary Computation Journal*, 10(4):371–395.

Eberhart, R. and Shi, Y. (1998). Comparison between genetic algorithms and particle swarm optimization. In *Proceedings of the Seventh Annual Conference on Evolutionary Programming*, pages 611–619.

Eberhart, R. C., Simpson, P., and Dobbins, R. (1996). *Chapter 6*, pages 212–226. AP Professional, San Diego, CA.

Hansen, N. and Ostermeier, A. (1996). Adapting arbitrary normal mutation distributions in evolution strageties: The covariance matrix adaptation. In *Proceedings of the IEEE International Conference on Evolutionary Computation*, pages 312–317.

Hansen, N. and Ostermeier, A. (2000). Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation Journal*, 9(2):159–195.

Higashi, N. and Iba, H. (2003). Particle swarm optimization with gaussian mutation. In *Proceedings of the IEEE swarm intelligence sysmposium 2003*, pages 72–79.

Juang, C.-F. (2004). 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.

Kennedy, J. (1997). The particle swarm: Social adaptation of knowledge. In *Evolutionary Computation, 1997., IEEE International Conference on*, pages 303–308.

Kennedy, J. (2003). Bare bones particle swarm. In *Proceedings of IEEE International Conference on Evolutionary Computation*, pages 80–87.

Kennedy, J. and Eberhart, R. (1995). Particle swarm optimization. In *Proceedings of Conference on Evolutionary Computation (CEC)*, pages 1942–1948.

Lovbjerg, M., Rasmussen, T. K., and Krink, T. (2001). Hybrid particle swarm optimizer with breeding and subpopulations. In *Proceedings of GECCO*.

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

Ozcan, E., Cad, S., Sok, T., and Mohan, C. K. (1999). Particle swarm optimization: Surfing the waves. In *Proceedings of the Congress on Evolutionary Computation*, pages 6–9. IEEE Press.

Padhye, N., Branke, J., and Mostaghim, S. (2009a). Empirical comparison of MOPSO methods: Guide selection and diversity. In *Proceedings of CEC*, pages 2516–2523.

Padhye, N., Mohan, C. K., Mehrotra, K. G., and Varshney, P. (2009b). Sensor selection strategies for networkds monitoring toxic chemical release. In *Poceedings of Sensor Networks Applications (SNA)*.

Pant, M., Thangaraj, R., and Abraham, A. (2007). *A New PSO Algorithm with Crossover Operator for Global Optimization Problems*, pages 215–222. Springer-Verlag, Berlin /Heidelberg.

Ratnaweera, A., Halgamuge, S. K., and Watson, H. C. (2004). Self-organizing hierarchical particle swarm optimizer with time-varying acceleration coefficients. *Evolutionary Computation, IEEE Transactions on*, 8(3):240–255.

Reklaitis, G. V., Ravindran, A., and Ragsdell, K. M. (1983). *Engineering Optimization Methods and Applications*. New York : Wiley.

Reyes-Sierra, M. and Coello, C. A. C. (2006). Multi-objective particle swarm optimizers: A survey of the state-of-the art. *International Journal of Computational Intelligence Research*, 2(3):287–308.

Schwefel, H.-P. (1995). *Evolution and Optimum Seeking*. New York: Wiley.

Shi, Y. and Eberhart, R. (1998a). A modified particle swarm optimization. In *Proceedings of the IEEE international conference on evolutionary computation*, pages 69–73.

Shi, Y. and Eberhart, R. (1998b). Parameter selection in particle swarm optimization. In *Proceedings of the 7th International Conference on Evolutionary Programming VII*, volume 1447, pages 591–600.

Storn, R. and Price, K. (1997). Differential evolution – A fast and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11:341–359.

Tawdross, P. and Koenig, A. (2006). Local parameters particle swarm optimization. In *Hybrid Intelligent Systems, 2006. HIS '06. Sixth International Conference on*.

Zhang, W. and Xie, X. (2003). DEPSO: Hyrbid particle swarm with differential evolution operator. In *IEEE international conference on systems, man and cybernetics (SMCC)*, volume 3410, pages 3816–3821.