Unified Approach In Evolutionary Computation For DE Enhancement

Nikhil Padhye[1], Piyush Bhardawaj[2], and Kalyanmoy Deb[2]

[1] Department of Mechanical Engineering
Massachusetts Institute of Technology, MA 02139, USA
npdhye@mit.edu
[2] Department of Mechanical Engineering
Indian Institute of Technology Kanpur
PIN 208016, Uttar Pradesh, India
{piyush,deb}@iitk.ac.in

Abstract. Only a few attempts in past have been made in adopting a unified outlook towards different paradigms in Evolutionary Computation (EC). The underlying motivation of these studies was aimed at gaining better understanding of evolutionary methods, both at the level of theory as well as application, in order to design efficient evolutionary algorithms for solving wide-range of complex problems. However, the past descriptions have either been too general or sometimes abstract in issuing a clear direction for improving an evolutionary paradigm for a task-specific. This paper recollects the 'Unified Theory Of Evolutionary Computation' from past and investigates four steps – *Initialization*, *Selection*, *Generation* and *Replacement*, which are sufficient to describe traditional forms of *Evolutionary Optimization Systems* such as Genetic Algorithms, Evolutionary Strategies, Evolutionary Programming, Particle Swarm Optimization and Differential Evolution. Then, a relatively new evolutionary paradigm, Differential Evolution, is chosen and studied for its performance on a set of unimodal problems. Discovering DEs inability as an efficient solver, DE is reviewed under 'Unified Framework' and functional requirements of each step are evaluated. Targeted towards enhancing the DE's performance, several modifications are proposed through borrowing of operations from a benchmark solver G3-PCX. Success of this exercise is demonstrated in a step-by-step fashion via simulation results. The *Unified Approach* is highly helpful in understanding the role and re-modeling of DE steps in order to efficiently solve unimodal problems. In an avalanching-age of new methods in EC, this study hints a direction for advancing EC methods by undertaking a collective outlook and an approach of concept-sharing.

1 Introduction

Much of the early research and development in evolutionary based computational methods occurred independently without any interaction(s) among various groups [17]. It was around late 1980s and early 1990s when the confluence of these paradigms began, which eventually led to the agreement on the term "Evolutionary Computation". Succeeding interactions among the evolutionary computation researchers led to the breeding of new algorithms, namely, Genetic Programming (GP), messy GAs, Samuel, CHC, Genocoop, Genitor, etc. The period afterwards marked the development of two major meta-heuristic techniques for optimization: Particle Swarm Optimization and Ant Colony Optimization, and the adaption of Genetic Annealing algorithm giving birth to Differential Evolution (DE) – all now being tagged under Evolutionary Computation and collectively referred to as Evolutionary Algorithms (EAs). In fact, the field of Evolutionary Computation nowadays includes and ties closely with self-organizing systems [18], artificial life [22], Memetic and Cultural Algorithms [20], Harmony Search [2], Artificial Immune Systems [4], and learn-able evolution model [1]. It is fair to state that the frontiers of EC reach far than ever expected but the growth in different EC areas continues without much interdependence. To educate oneself on popular EA (and related) paradigms reader is referred to following standard resources [12],[17],[19],[5].

In-spite of advances in different EA paradigms there have been only a few attempts, if not the lack of interest, in search for a framework which is capable of conceptualizing any given evolutionary algorithm and explain its behavior. A plausible approach could be to think about decomposing an EA into key standard components (where standard components assembled together represent the EA). Then, by understanding the role of each component individually and associated interactions amongst the components; insights into an EA's performance can be drawn. If successful, such a strategy shall provide a generic representation for different evolutionary paradigms itself and we shall refer this to as *Unified Approach* towards EC. Two notable attempts in past adopting the *Unified Approach* have been made in [17], [6]. However, the more challenging question – How to use *Unified Approach* to improve an evolutionary algorithm? or any procedural demonstration stays undressed.

This paper samples concepts from past two studies ([17], [6]) and presents a *Unified Approach* for evolutionary algorithms (like GAs, ES, EP, PSO and DE); in context to real-parameter optimization for unimodal problems. Then the primary focus is shifted on the performance of standard DE algorithm compared against a benchmark genetic algorithm G3-PCX [9]. After discovering inefficient DE performance, *Unified Approach* is adopted in analyzing and altering key DE steps. The DE steps are modified by borrowing ideas from G3-PCX and gradual improvement in performance is noted. Through a series of seamless modifications DE's performance is enhanced to an extent where it becomes comparable to the benchmark results, and the modified DE algorithm becomes to equivalent to G3-PCX. The procedure highlights that how using the *Unified Approach* frame-work the similarities and differences between two algorithms can be understood, and based on the functional requirements how the key algorithmic steps are modified to meet the desired performance levels. The study demonstrates that as the individual steps of two algorithms become similar; the performance-gap is lessened and when the algorithmic equivalence is achieved the performances match. It also becomes evident that in order to understand and improve the performance of an EA its design should be noted and the non-linear interactions among different operators along with their synergistic effect should be considered.

The rest of the paper is structured as follows: Section 2, presents an *Unified Framework* for evolutionary optimization algorithms and discusses several EA paradigms based on this framework. Section 3, provides an introduction to unimodal test problems chosen in this paper and description on the experimental methodology. The performance of benchmark algorithm, G3-PCX, specifically designed to solve unimodal problems is also presented here. Sections 5 and 6, detail the performance of standard DE and its several variants oriented towards a performance improvement. Details on several proposed strategies for modifications are also provided here, along with the reasoning for their effectiveness or ineffectiveness. Section 7, compares the performance of standard DE along with its variants and benchmark algorithms on problems with large number of variables. Finally, Section 8 concludes the paper and hints on the direction for the future work.

2 Recollection of Unified Framework For Evolutionary Algorithms – A Modest Description

The most notable *breaking new ground* attempt in adopting *A Unified Approach* towards Evolutionary Computation is made in [17], serving the goal of presenting an integrated view of Evolutionary Computation. This paper takes a step forward in demonstrating – How the *Unified Approach* can be applied in better understanding (and thereby improving) an EA paradigm?

Fig. 1. Evolutionary Optimization System based on *EV-OPT* proposed by [17]

```
Randomly generate the initial population of M individuals
(using a uniform probability distribution over the entire
geno/phenospace) and compute the fitness of each individual.

Do until a defined stopping criterion is met:
  – Select member(s) of the current population to be the parent(s).
  – Use the selected parent(s) to produce offspring(s).
  – Select member(s) of the population to Die.
End Do

Return the individual with best fitness value.
```

The *Unified Approach* was introduced in [17] where the author outlined a most general form for *Evolutionary Optimization System(EOS)* based on the Darwinian evolutionary system. The key steps of such an *EOS* are shown in

Figure 1. The *EOS* has been assumed to be constant in population size and the optimization task being that of minimization. The key steps in *EOS* are: (1)*Initialization* – of the population randomly, (2) *Selection* – of the individual(s) from the population to act as parent(s), (3) *Generation* – Creation of offspring(s) from the selected parent(s), and (4)*Replacement* – Selection of individuals(s) to survive for the next generation. After the *Initialization*; *Selection*, *Generation* and *Replacement* are repeated iteratively till a pre-specified termination criterion is met. Although detailed description on each step are required before *EOS* can be simulated, but just *a few steps* procedure as above is sufficient to represent major EA paradigms for optimization. It is important to guard the reader from the fact that the description of an *EOS* provided here may not be suited for representing evolutionary methods which involve adaptation and self-adaptation of the parameters etc. However, the purpose unified frame-work is not so much to represent each and every existing evolutionary algorithm or their artifacts, but more so in identifying a common reference for different paradigms. *EOS*

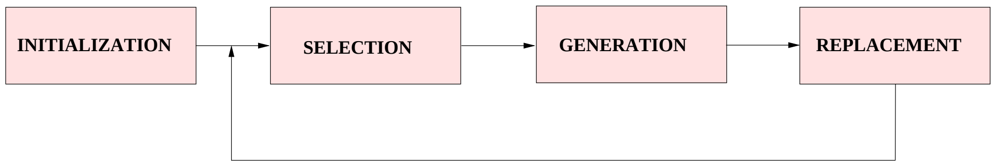


Fig. 2. For major Steps in an *EOS*

described above requires an additional elaboration on population management i.e. how do offsprings compete for survival. Two popular ways are: (a) *Steady State* – or *incremental model*, implying that offsprings are produced one at a time and immediately compete for the survival i.e. if the fitness of child is better than the parent selected, the child survives or vice-versa, or (b) *Generational* – or *batch model*, implying that entire batch of child population is created and then there is competition for survival. The notation adopted in this paper to represent evolutionary systems has been borrowed from [17] – Two populations are maintained: one of size m for parents and second of size n for offsprings (now the system being represented as *EOS(m,n)*). In *EOS(m,n)*, n offsprings are created from the parent population of size m and then each child competes for space in the parent population. For, a special case, $n = 1$ we arrive at steady state model, and any value of $n > 1$ symbolizes generational model. *EOS*s discussed in this paper are associated with real parameter optimization where solutions are represented as vectors of real parameter decision variables.

Next, we shall discuss popular EA paradigms as instances of *EOS(m,n)*. This exercise shall serve following two purposes (i) provide a gateway to understand and express standard evolutionary methods in an EOS(m,n) form, and (ii) recollect popular operators being employed in different evolutionary methods. Such a collection becomes highly useful when operators and ideas have to be borrowed and applied from one evolutionary method to another.

2.1 Real-parameter Genetic Algorithms as EOS(m,n)

In real-parameter genetic algorithm (rGA), the population is randomly initialized, and a set of genetic operations (selection, recombination, mutation, and elite-preservation) are performed to create a new population in an iterative manner. Most rGAs differ from each other in their genetic operations and/or replacement plan. Popular operators and/or replacement plan are summarized as follows:

Selection – Role of this operator is to *prefer better solutions to worse ones.* The selection operators can be divided into two basic categories: *deterministic* and *stochastic* selection methods. With *deterministic* methods, each individual in the selection pool is assigned a *fixed number* that corresponds to the number of times they will be selected. For e.g., each individual can be selected exactly once for creation of an offspring, or individuals are sorted according to their fitness values and top few are selected (*truncation selection*). On the other hand, *stochastic* selection methods assign a probability to each individual according to which it is selected. Common examples of such selection methods are Uniform, Fitness-proportional, Linear ranking and binary tournament, Nonlinear ranking and tournaments with tournament size greater than two, etc.

Generation: Crossover and Mutation

Linear Crossover, Blend Crossover, Arithematic, Simulated Binary Crossover, Fuzzy Recombination Operator, Unfair Average Crossover, Simplex Crossover, Unimodal Normally Distributed Crossover, Fuzzy Connectives Based Crossover, Parent Centric Crossover, etc.

Random Mutation, Polynomial Mutation, Non-Uniform Mutation, Normally Distributed Mutation, etc.

Details on above operations can be found in [7].

Replacement – can be carried out based on either steady state or generational model, though it is worthwhile to mention that standard genetic algorithms ([16],[17],[12]) utilized a generational model in which parents survived for exactly one generation and completely replaced by their offsprings. Thus, standard GAs can be thought of as EOS(m,m).

2.2 Evolutionary Strategies as EOS(m,n)

Evolutionary Strategies (ES) [23] have been fundamentally different from binary Genetic Algorithms principally in two ways: (1) ESs used real parameter values, and (2) early ESs did not have any crossover-like operators (i.e. non-recombanative, though later recombanative ESs were also proposed). Two standard approaches in ES, $(\mu + \lambda)$-ES and (μ, λ)-ES can be represented in form of *EOS(m,n)* as follows:

$(\mu + \lambda)$-ES as *EOS(μ, λ)*:

Selection – Uniform selection.

Generation – Normally Distributed Mutation.

Replacement – According to the Steady State model i.e the *best fitness* individual from parent and offspring is preserved. Since offsprings compete with the parents directly, $(\mu + \lambda)$-ES is an elitist algorithm.

(μ, λ)-ES as *EOS(μ, λ)*:

Selection – Uniform selection.

Generation – Normally Distributed Mutation.

Replacement – According to the Generational model, in which batch of λ offsprings is created from μ parents (with $\lambda > \mu$), and top μ offspring in terms of *best fitness values* form the parent population for next generation. (μ, λ)-ES is a non-elitist algorithm.

Recombanative Evolutionary Strategies $(\mu/\rho + \lambda)$-ES and $(\mu/\rho, \lambda)$-ES can also be represented in form of *EOS(m,n)* by modifying *Selection* and *Generation* operations in $(\mu + \lambda)$-ES and (μ, λ)-ES, respectively.

2.3 Evolutionary Programing as EOS(m,m)

Evolutionary Programming (EP) is a mutation driven evolutionary algorithm applicable to real-parameter optimization [11]. EP is similar to ES in the fact that normally distributed mutations are performed in both algorithms. In standard EV, each population member is deterministically selected to create an offspring (whether good or bad is a different issue). If we consider an *EOS(m,n)*, with parent and offspring populations of equal size ($m = n$), then EP can be represented as *EOS(m,m)* with following operations:

Selection – Deterministic selection i.e. each individual is selected as a parent.

Generation – Normally distributed mutation with zero mean and fitness-function dependent variance.

Replacement – According to the Generational model, $2m$ parents and children are combined and only m individuals with *best fitness values* are preserved for next generation.

The EP implements a much stronger elitist survival scheme in which only the top 50% survive (which is also often responsible for its faster convergence).

2.4 Particle Swarm Optimization as EOS(m,m)

Particle Swarm Optimization algorithms consist of several particles, flying in the search space, and have been thought to simulate the behavior of organisms like those in the bird flocks or fish schools. Each particle in a standard PSO maintains a *personal best* and a *global best* is maintained for the entire swarm. Particles update their locations under the influence of their own *velocities*, *personal best* and *global best*. In an earlier studies [10], the authors presented an archive-based evolutionary algorithm which was equivalent to Particle Swarm Optimization(PSO). Using that analogy, PSO algorithms can represented as *EOS(m,n)* with $m = n$, as follows:

Selection – Deterministic selection i.e. each particle (or individual) is selected as a parent.

Generation – Crossover-like operation (same as particle move equation), and random mutation (same as "turbulence" in PSO).

Replacement – According to the Generational model, entire offspring population replaces the parent population (along with it *velocity*, *personal best* and *global best* are updated).

Steady State versions of standard PSO were found to perform comparatively better [10].

So far we have reviewed rGAs, ES, EP, and PSO under *Unified Approach* plan and it is clear that at the level of abstraction they are "similar". It is worth asking a fundamental question – Why different EOSs exist and what purpose do they serve? From a practitioner's perspective each evolutionary optimization paradigm has features and properties which makes it more preferred over the other. The choice of an *EOS* for an application is often contextual and depends on how easily an *EOS* can be adapted for the given task at hand. Obviously there is no statutory winner or looser. The goal of this study is to substantiate this very fact while highlighting how do the properties of an *EOS* (i.e. the key steps) affect its performance in a given scenario and how can appropriate modifications be introduced to achieve desired results. We shall return to this point later, but next we provide a detail description of DE as an *EOS*.

2.5 Differential Evolution as EOS(m,m)

Differential Evolution (DE) algorithm has emerged as a very competitive form of evolutionary computing for a decade now. The main goal of this study is to develop a thorough understanding of DE algorithm as an *EOS* and then systematically exploit this understanding in improving DE's performance. Since majority of simulations presented in this study are based either on standard DE or its modified versions, we provide its pseudo code in Figure 3.

Fig. 3. Standard Differential Evolution *(DE/best/1/exp)* and its decomposition, borrowed from [19].

```
Input DE parameters: scale factor (F), crossover-rate (Cr) and population size (M)
Initialization
 Randomly Generate the initial population of M individuals in the defined region
 and compute the fitness of each individual.
Set Generation Counter $t = 1$
Do until a defined stopping criterion is met:
   For $i = 1$ to $M$
    Selection
     – Choose, $i^{th}$ individual ($X_{i,t}$), two random individuals ($X_{r1,t}$,$X_{r2,t}$), and best member
     in the population at previous ($t-1$) generation ($X_{best}$) as parents
    Generation
      (a) Create $i^{th}$ Donor Vector:
      $V_{i,t} = X_{r3} + F \cdot (X_{r1,t} - X_{r2,t})$
      (b) Create $i^{th}$ Trial Vector:
      $U_{i,t}$=CombineElements($X_{i,t}, V_{i,t}$) // with probability $CR$
    Replacement
     If (Fitness($U_{i,t}$)$\leq$Fitness($X_{i,t}$))
     Then $X_{i,t+1} = U_{i,t}$
     Else $X_{i,t+1} = X_{i,t}$
   End For
   Update($X_{best}$)
 $t = t + 1$
 Update($P_{t+1}$)
End Do
Return the individual with best fitness value.
```

The standard DE presented here is exactly same as those in "DE/best/1/exp" [19]. The population is scanned serially and a child is created corresponding to an individual using four parents (the individual itself which we shall refer to as the base or index parent, *best fitness* individual from the previous generation and two randomly chosen population members). First a donor vector ($V_{i,t}$) is created (step a) and then a trial vector ($U_{i,t}$) is created (step b) by stochastically combining elements from $X_{i,t}$ and $V_{i,t}$. This combination is commonly done using an exponential distribution with crossover factor of CR. If the newly created

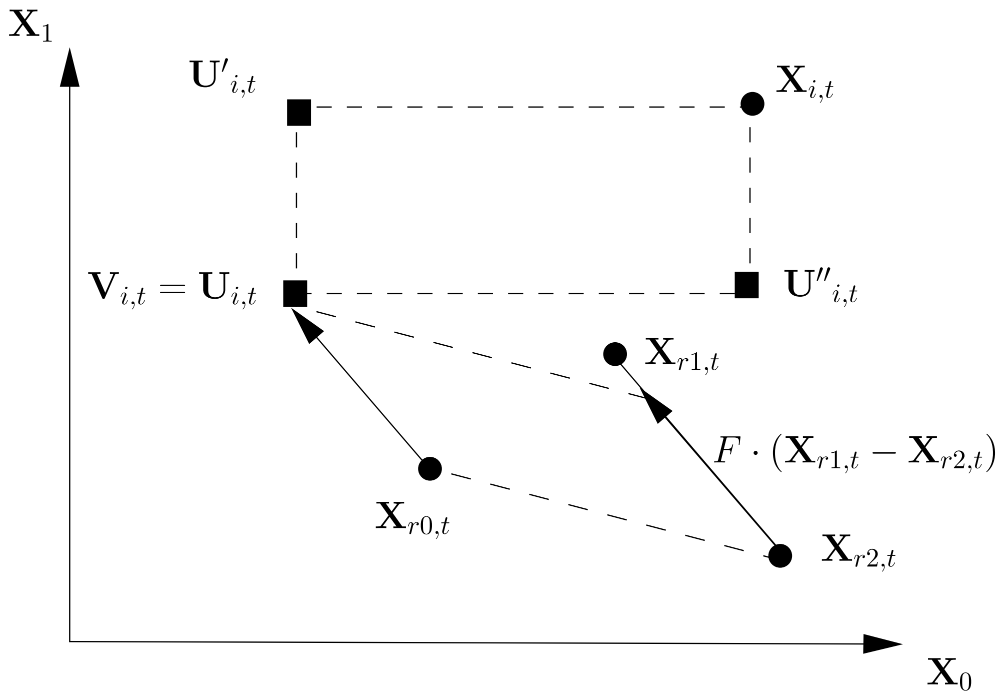


Fig. 4. DE update rule

child $U_{i,t}$ is *better* compared to $X_{i,t}$ then $U_{i,t}$ is stored for updating $X_{i,t+1}$. It should be noted carefully that $X_{i,t}s$ are updated to $X_{i,t+1}s$ after entire set of $U_{i,t}s$ are created. Once the population is updated, the generation counter is incremented and termination criteria is checked.

Following properties of standard DE are worth noting: (i) There is 'elitism' at an individual level i.e. if the newly created trial vector $U_{i,t}$ is inferior compared to the individual then individual is preserved as a child for the next generation and $V_{i,t}$ is ignored. (ii) The algorithm follows a generational model i.e. the current population is updated only after the entire offspring population is created.

3 Test Suite – Unimodal Problems

In this study, we have considered unimodal problems (having one optimum solution) or problems having a few optimal solutions. Such test problems allow us to test following two properties of an algorithm: (i) it's ability to progress towards the optimal region, and (ii) it's ability to find the optimum within a specified precision after reaching the optimal basin.

A previous study [9] 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 three unimodal problems (which we have also adopted in this paper):

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

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

$$F_{\mathrm{ros}} = \sum_{i=1}^{n-1} \left(100(x_i^2 - x_{i+1})^2 + (x_i - 1)^2 \right) \quad \text{(Generalized Rosenbrock's function)} \tag{3}$$

The first two problems described above have a minimum at $x_i^* = 0$ with $F^* = 0$ and the third problem has a minimum at $x_i^* = 1$ with $F^* = 0$. For all three problems the number of variables (n) was chosen to be 20.

In order to study the algorithm's ability to progress towards the optimal region, for all the problems the population is initialized away from the optima such that $x_i \in [-10, -5]$ for all i. However, in subsequent generations solutions are not confined in this region. Initialization of population around the known optimum does test an algorithm's ability to approach the optimal region, rather in most cases it tests the algorithm's ability to converge precisely within the optimal solution.

An efficient algorithm, starting from an initial guess solution must first approach an optimal region and then concentrate in the region to find the optimal solution within a desired precision. In this study, we investigate both aspects of an algorithm by considering two evaluation criteria: First, after randomly initializing the population 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 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.

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.

In the previous study [9] G3-PCX algorithm was found to be an overall winner on the set of unimodal problems described here. Therefore, we review the properties of this algorithm next and subsequently borrow them for enhancing the DE's performance.

4 Generalized Generation Gap based Genetic Algorithm (G3-PCX) – Key Features and Functions

The study in [9] 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 [9]. One iteration of algorithm is described as follows:

Selection: From the population P, select the best solution as a parent and choose $\mu - 1$ other parents randomly.
Generation: Generate λ offspring from the chosen μ parents using the PCX recombination scheme.
Replacement: Choose two parents p_a and p_b at random from the population P. 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. The newly created child solutions don't have to wait (unlike generational models) and can become parents as quickly as after two function evaluations. For solving unimodal problems, this property provides a selection pressure for good solutions.
– It uses an elite-preservation operator, since children are compared against parents before getting accepted into the population.
– It uses a recombination operator that creates child solutions around the globally best solution. For solving unimodal problems in a computationally fast manner, this operation helps the current best solution to move towards the optimum location.

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

Table 1. Benchmark Performance of G3-PCX, results as reported in [9].

	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

comparison with a CMA-ES [13, 14], a couple of self-adaptive evolution strategies [24, 3], differential evolution [25] and a quasi-Newton classical method [21], the number of function evaluations reported in the above table were found to be minimum. In following sections sections, we shall present the simulation results and show comparisons based on S_1 and S_2 metrics. The target goal would be to match the performance as shown in Table 1.

5 Performers from the DE family

Storn and Price proposed a total of ten different schemes for DE [19]. These schemes differed only in their *Generation* step. Guidelines for choosing DE parameters, F and CR, were also provided for these schemes. In particular, there were 5 different ways of creating the trial vector (*Step a*, Figure 3), and 2 ways of combining the elements from donor vector and trial Vector (*Step b*, Figure 3), thereby leading to a total of 10 ways of generating a new solution. No single generation scheme described in family turned out to be best for all kinds of problems.

While selecting the *Generation* scheme in this paper, we tested all 10 DE schemes (with their recommended parameter settings) on three test problems, and noted their performances w.r.t. S_1 and S_2 metrics. The results for different *generation schemes* (tagged as 'strategy number') are shown in Table 2. *Strategy 1* and *Strategy 6* turn out to be best performers. *Strategy 6* clearly beats *Strategy 1* on F_{elp} (w.r.t. S_1 and S_2) and F_{ros} (w.r.t. S_2). Arguably the performance of *Strategy 1* is competitive (or marginally inferior) compared to *Strategy 6*. It is worth mentioning that for *Strategy 1* the solutions are generated around the *best* fitness individual of the previous iteration. This bears similarity with G3-PCX algorithm where the *current best* is always involved in creation of a new solution, and leads us to investigate further.

We performed a parametric study on M, CR and F for *Strategy 1* and found an overall improved performance, as shown in Table 3. $M = 50$, $CR = 0.95$ and $F = 0.7$ were found as optimal values with respect to all the three test problems. Under same optimal parameter settings *startegy 6* did not perform well on F_{sch} and F_{ros} functions, as shown in Table 4. Therefore in remainder of this paper, DE with *Strategy* 1 is employed for simulations and shall be referred to as standard DE.

6 Functional Analysis of DE components and Modifications

The goal of this section is to understand and then modify the key DE steps. Firstly, we consider standard DE algorithm and analyze its key features. Then, we modify the key DE steps (*Selection*, *Generation* and *Replacement*) one-by-one and note a variation in the performance. In particular, key steps are altered with operations such as *Parent-Child Comparison*, *Best Update*, *Steady State Update*, *Random and Tournament Selection*, *Random or Serial Parent Replacement* and *Mutation*. Most of the modifications introduced in standard DE are motivated

Table 2. "DE/best/1/exp" [19], with different DE strategies.

	F_{elp}			F_{sch}			F_{ros}		
	Best	Median	Worst	Best	Median	Worst	Best	Median	Worst
Strategy 1, $F = 0.7$, $Cr = 0.5$, $NP = 50$									
S_1 with 10^{-1}	10,600	11,450	12,200	**61,850**	76,700	**90,350**	577,600(49)	760,550	886,700
S_2 with 10^{-20}	53,550	55,050	56,000	494,250	**522,000**	**541,450**	3.3e-09 *DNC*	7.1e-06 *DNC*	2.5e-01 *DNC*
Strategy 2, $F = 0.7$, $Cr = 0.5$, $NP = 50$									
S_1 with 10^{-1}	13,600	15,050	15,900	99,100	120,050	135,950	656,850	681,250	778,750
S_2 with 10^{-20}	70,150	72,100	73,400	844,250	888,100	917,500	3.7e-11 *DNC*	6.5e-09 *DNC*	6.5e-07 *DNC*
Strategy 3, $F = 0.85$, $Cr = 1.0$, $NP = 50$									
S_1 with 10^{-1}	448 *DNC*	1,460 *DNC*	4,610 *DNC*	37.9 *DNC*	258 *DNC*	813 *DNC*	25,600 *DNC*	372,000 *DNC*	2,280,000 *DNC*
Strategy 4, $F = 0.7$, $Cr = 0.5$, $NP = 50$									
S_1 with 10^{-1}	15,350	16,650	17,650	146,900	164,100	184,900	2.99 *DNC*	5.87 *DNC*	8.12 *DNC*
S_2 with 10^{-20}	81,500	83,750	85,400	3.75e-20	2.48e-19	1.48e-18	2.99 *DNC*	5.87 *DNC*	8.12 *DNC*
Strategy 5, $F = 0.7$, $Cr = 0.5$, $NP = 50$									
S_1 with 10^{-1}	18,750	19,750	20,600	190,300	215,550	239,150	546,050(18)	930,800	999,150
S_2 with 10^{-20}	97,200	99,050	100,850	2.97e-14 *DNC*	1.22e-13 *DNC*	6.9e-13 *DNC*	4.3e-03 *DNC*	0.25 *DNC*	4.59 *DNC*
Strategy 6, $F = 0.7$, $Cr = 0.5$, $NP = 50$									
S_1 with 10^{-1}	**6,400**	**7,150**	**7,700**	62,700	**76,400**	96,250	**54,150(47)**	**127,000**	**149,950**
S_2 with 10^{-20}	**35,750**	**37,550**	**39,250**	**485,400**	527,300	581,750	**245,850(47)**	**340,850**	**377,300**
Strategy 7, $F = 0.7$, $Cr = 0.5$, $NP = 50$									
S_1 with 10^{-1}	20,300	21,550	23,050	527,300	643,300	726,600	294,600	309,450	324,700
S_2 with 10^{-20}	107,850	111,150	114,750	1.40e-04 *DNC*	6.87e-04 *DNC*	3.65e-03 *DNC*	1.18e-16 *DNC*	6.97e-16 *DNC*	2.43e-14 *DNC*
Strategy 8, $F = 0.85$, $Cr = 1.0$, $NP = 50$									
S_1 with 10^{-1}	448 *DNC*	1,460 *DNC*	4,610 *DNC*	37.9 *DNC*	256 *DNC*	814 *DNC*	25,700 *DNC*	372,000 *DNC*	2,280,000 *DNC*
Strategy 9, $F = 0.7$, $Cr = 0.5$, $NP = 50$									
S_1 with 10^{-1}	32,950	36,000	38,600	1.54e-01 *DNC*	6.57e-01 *DNC*	1.99 *DNC*	460,400	521,700	663,150
S_2 with 10^{-20}	180,500	188,600	195,250	1.54e-01 *DNC*	6.57e-01 *DNC*	1.99 *DNC*	6.26e-08 *DNC*	5.72e-07 *DNC*	9.14e-06 *DNC*
Strategy 10, $F = 0.7$, $Cr = 0.5$, $NP = 50$									
S_1 with 10^{-1}	72,600	76,750	82,850	6.31e+01 *DNC*	1.46e+02 *DNC*	2.81e+02 *DNC*	1.06e-01 *DNC*	2.10e-01 *DNC*	6.42e-01 *DNC*
S_2 with 10^{-20}	386,300	397,800	408,450	63.1 *DNC*	146 *DNC*	281 *DNC*	0.10 *DNC*	0.210 *DNC*	0.642 *DNC*

Table 3. Standard DE with *Strategy 1*, “DE/best/1/exp” [19], with optimal parameter settings of $F = 0.7$, $CR = 0.95$, $M = 50$.

	F_{elp}			F_{sch}			F_{ros}		
	Best	Median	Worst	Best	Median	Worst	Best	Median	Worst
S_1 with 10^{-1}	**6,100**	**6,600**	**7,200**	**7,600**	**9,000**	**11,200**	**21,050(43)**	**29,500**	**33,850**
S_2 with 10^{-20}	**31,700**	**33,550**	**35,100**	**48,050**	**51,100**	**55,200**	**55,400(43)**	**63,350**	**69,350**

Table 4. Standard DE with *Strategy 6*, “DE/best/1/bin” [19], with parameter settings of $F = 0.7$, $CR = 0.95$, $M = 50$.

	F_{elp}			F_{sch}			F_{ros}		
	Best	Median	Worst	Best	Median	Worst	Best	Median	Worst
S_1 with 10^{-1}	**4,400**	**5,900**	**9,600**	8,800	11,500	15,700	32550(43)	113,400	255,600
S_2 with 10^{-20}	**25,750**	**28,950**	**35,950**	57,000	70,400	79,000	81,000(35)	172,950	319,700

Table 5. Comparing Standard DE with and without $Parent - Child\ Comparison$, $Best\ Update$ and $Steady\ State$ operations. $F = 0.7$, $CR = 0.95$, $M = 50$.

	F_{elp}			F_{sch}			F_{ros}		
	Best	Median	Worst	Best	Median	Worst	Best	Median	Worst
Standard DE without $Parent - Child\ Comparison.$									
S_1 with 10^{-1}	31,450	66,450	152,050	70,300	148,550	297,050	253,000(35)	660,750	861,750
S_2 with 10^{-20}	329,450	403,350	540,850	862,500(11)	921850	990,600	5.12e-10	1.55e-03	3.99
Standard DE without $Parent - Child\ Comparison + Best\ Update.$									
S_1 with 10^{-1}	46,200	68,550	119,200	70,900	147,350	36,550	172,550(34)	527,350	867,050
S_2 with 10^{-20}	288,350	407,300	517,800	746,700(10)	891,100	987,700	1.40e-10	1.40e-05	3.99
Standard DE without $Parent - Child\ Comparison + Best\ Update\ +\ Steady\ State\ Update$									
S_1 with 10^{-1}	30,850	58,300	136,300	94,350	172,600	401,400	313,400(38)	700,450	987,350
S_2 with 10^{-20}	249,650	379,850	572,150	823,750(15)	925,800	995,800	3.90e-07 *DNC*	5.47e-03 *DNC*	4.097 *DNC*

from G3-PCX and show a gradual trend of improving performance. Taking hints from these set of modifications, an efficient *Elitist DE* based on a generational model is also proposed. When the best of DE results are unable to compete with the benchmark G3-PCX results, we invoke PCX operation in DE and call name the algorithm as DE-PCX. After fine tuning of parameters, DE-PCX performs as good as G3-PCX and ultimate goal of this study is achieved.

6.1 Anatomy of Standard DE

'Elitism', or preservation of *good individuals* in an EA procedure is usually a part of *Replacement* step and often results in a performance improvement. One of the most noticeable features of standard DE is that it employs *elitism* at an individual level i.e. a child individual is compared with its *base parent* (the parent at the index corresponding to which child was created) and only the better of the two is retained in the next generation. The operation of standard DE is shown in Figure 5(a), where parent population P_t is scanned serially from the top and corresponding to each *index parent* a child is created and stored in the offspring population Q_t. The parent population for next generation $(t+1)$ is created by comparing same index individuals from P_t and Q_t and placing the better of the two at corresponding index in Q_{t+1}. We shall refer to this comparison as *Parent-Child Comparison.*

Before proceeding further we draw a comparison between standard DE and G3-PCX and introduce following concepts to characterize the key steps in DE:

Best Update – The *Generation* scheme (Step a) in standard DE involves creation of a child around $X_{best,t-1}$. This approach of creating solutions around the *best* is particularly useful in solving unimodal problems. The benchmark algorithm G3-PCX successfully exploits this property. But, a major difference in the *Generation* step of G3-PCX and the standard DE lies in the fact that former uses the *current best* location in the population, whereas DE utilizes the *previous generation's best.* We shall incorporate this feature in standard DE by using X_{best} instead of using $X_{best,t-1}$, where X_{best} indicates the best known location so far. This is achieved by checking and updating X_{best} after every new child creation. Since X_{best} represents the globally known best location, we shall refer to this strategy as **Best Update**.

Steady State Update – Another basic difference between the benchmark algorithm G3-PCX and standard DE lies in their steady state and generational models, respectively i.e. the *Replacement* step. In standard DE, firstly all the offsprings are created and stored separately. Then, at the end of the current cycle the parent population is compared and replaced with child population. Thus, each offspring has to wait till next generation before it can be selected as the index parent. We propose the steady state version of standard DE (and call this technique as **Steady State Update**) in which as soon as a child is created it is compared with its index parent and checked for replacement. Figure 5(b) shows this process.

Now, we make the first attempt in understanding the importance of elitism through *Parent-Child Comparison.* The *Replacement* step of standard DE is modified by always accepting the newly created child (unlike before where the child is accepted only if it was better). The DE without the $Parent - Child$ $Comparison$ resulted in a degraded performance in all three test problems with respect to both the metrics, indicating that elitism in DE by $Parent - Child$ $Comparison$ is a key feature for its better performance. Further, in absence of *Parent-Child Comparison*, we test *Best Update* and *Steady State Update.* Although *Best Update* shows a minor improvement in few cases but overall results reveal that compared to the standard DE all the variants are worse in performance as much as an order of magnitude in terms of number of function evaluations. Table 5 systematically compares the effects of removing $Parent - Child$ $Comparison$, adding *Best Update*, and testing the *Steady State Update.* Thus, we conclude that $Parent - Child\ Comparison$ is an important feature in the DE algorithm and in its absence other operations such as *Best Update* or *Steady State Update* are ineffective in making-up for the loss of 'elitism' which is brought by $Parent - Child\ Comparison$ as part of the *Replacement step.*

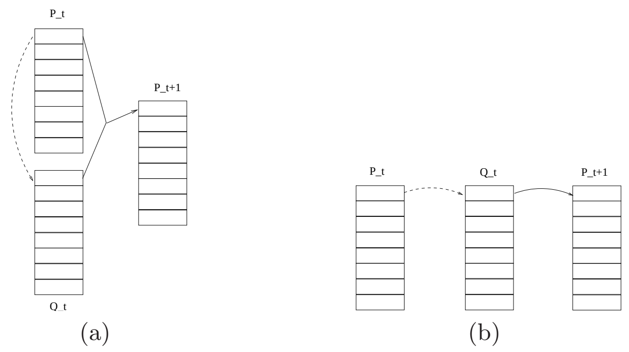


Fig. 5. (a) Population management in standard DE based on a generational model. Corresponding to each individual in parent population P_t a child is created and stored in child population Q_t. Parent population for next generation P_{t+1} is formed by selecting a better individual from corresponding indexes in P_t and Q_t. (b) *Steady State Update (Serial Parent Replacement)* in DE. As a child is created corresponding to an index parent it becomes a potential candidate for replacing the index parent (if the *Parent-Child Comparison* is not carried out then child always replaces the index parent; however if *Parent-Child Comparison* is carried out then only the better of the two is retained at the index).

Next, we investigate the role of *Selection* step and try following two selection schemes with standard DE: *Random* and *Tournament* (instead of the usual deterministic serial parent selection). *Random* selection probabilistically assigns equal opportunities to all members whereas *Tournament* selection allows greater opportunities to individuals with better fitness. It should be noted that here we are referring to selection of base parent; the other two members needed for child creation are still selected randomly. The results with alternate selection schemes are shown in Table 6 and indicate that the alternate selection schemes perform poorly compared to serial selection (Table 3). Poor performance of *Random* selection arises from too much variability in the parent selection (given the fact that other two parents in creation of the donor vector are also selected randomly, Figure 3), whereas *Tournament* selection leads to too high selection pressure on good individuals. Therefore, we conclude that deterministic serial selection finds a right balance in selecting parents for child creation. This also leads to an important realization that there is universally good *Selection* (or for that matter any other) operator for an evolutionary algorithm. The effectiveness of any operator is largely contextual and relies on the overall genetic make-up of the algorithm, decomposing an algorithm into key standard components is a promising strategy to better understand its behavior and suggest improvements.

Table 6. Standard DE with *Random* and *Tournament* selection, $CR = 0.95$, $F = 0.7$, $M = 50$.

	F_{elp}			F_{sch}			F_{ros}		
	Best	Median	Worst	Best	Median	Worst	Best	Median	Worst
Random Selection									
S_1 with 10^{-1}	14,250	16,700	20,050	15,950	20,400	24,600	50,800(38)	56,900	64,500
S_2 with 10^{-20}	58,350	64,900	72,100	116,000	122,000	132,700	136,750(38)	147,050	158,900
Tournament Selection									
S_1 with 10^{-1}	14,600	16,850	19,400	17,700	21,250	25,300	39,500(44)	52,950	66,650
S_2 with 10^{-20}	86,350	92,950	97,650	114,800	124,700	134,900	100,050(44)	114,850	132,300

Table 7. Standard DE + *Best Update*, $F = 0.7$, $CR = 0.95$, $M = 50$.

	F_{elp}			F_{sch}			F_{ros}		
	Best	Median	Worst	Best	Median	Worst	Best	Median	Worst
S_1 with 10^{-1}	**5,500**	**6,250**	**6,900**	**7,450**	**8,800**	**10,800**	**18,550(44)**	**24,350**	**29,000**
S_2 with 10^{-20}	**31,400**	**32,650**	**34,500**	**43,600**	**48,600**	**52,100**	**51,400(44)**	**56,750**	**62,400**

It is worthy to pause for moment and reflect on the path being followed. We set out by decomposing the DE into standard EOS steps. Then, the *Replacement* step is considered and its features are investigated. It is found that the effectiveness of DE lied in the *Parent-Child Comparison* of *Replacement step.* We also found that by resorting to the alternate *Generation step* i.e. (by creating solutions around X_{best} instead of X_{best-1}), no improvement was obtained. The *Steady State Update* instead of generational model was also ineffective without the *Parent-Child Comparison.* This leads to the conclusion that *Parent-Child Comparison* of the *Replacement step* working in conjunction with the serial *Selection* scheme has a dominant effect on DE's performance. Thus, the *Unified Approach* plan is helpful in understanding the role of each component individually and also their associated interactions. It is recommended that reader bears this thought in mind for remainder of the paper.

From Table 5 we recall that *Best Update*-standard DE-without- *Parent-Child Comparison* showed an improvement compared to standard DE-without- *Parent-Child Comparison* in a few cases. Therefore, here we only alter the *Generation* step of standard DE by invoking *Best Update* (while keeping the *Parent-Child Comparison.* The results are shown in Table 7 and an improved performance is obtained in all cases. It is obviously clear that creating the solutions around the current best individual i.e. X_{best} (instead of $X_{best,t-1}$) is a helpful strategy while solving unimodal problems.

Table 8. Standard DE + $Steady\ State\ Versions$, $CR = 0.95$, $F = 0.7$, $M = 50$.

	F_{elp}			F_{sch}			F_{ros}		
	Best	Median	Worst	Best	Median	Worst	Best	Median	Worst
Standard DE + *Steady State(Serial Parent Replacement)*									
S_1 with 10^{-1}	**5,300**	**6,000**	**6,850**	**7,200**	9,050	12,600	23,600(36)	28,950	35,850
S_2 with 10^{-20}	**28,750**	**29,650**	**32,200**	46,300	50,100	55,250	54,550(36)	60,950	70,850
Standard DE + *Steady State(Random Parent Replacement)*									
S_1 with 10^{-1}	**4,000**	**5,200**	**6,850**	**6,050**	**8,500**	11,450	20,750(40)	29,750	37,200
S_2 with 10^{-20}	**23,150**	**25,200**	**27,150**	**42,100**	**47,700**	54,350	53,050(40)	66,300	76,900

Since *Best Update* showed an improved performance, we also test for *Steady State Update* with standard DE. According to *Steady State Update* with standard DE, as soon as an offspring is created it is compared with a parent individual, and the if the offspring is better in terms of fitness it replaces the parent. An obvious choice for selecting a parent is to pick the index parent itself (referred to as *Steady State (Serial Parent Replacement)* strategy), shown in Figure 6. Alternately, an individual could be selected randomly from the parent population (referred to as *Steady State (Random Parent Replacement)* strategy) and compared for

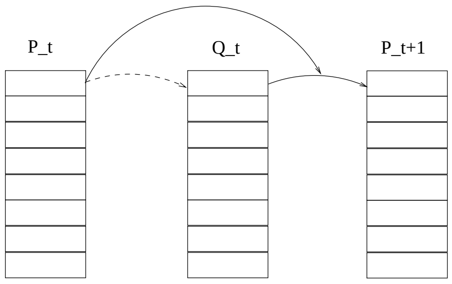


Fig. 6. Steady State Update (Serial Parent Replacement) in DE. As soon as a child is created corresponding an index parent it compared with the parent individual and replaces the parent if child is better.

replacement. Results for both the steady state update strategies are shown in Table 8 and indicate an improvement even over the standard DE with *Best Update* (Table 7) in some cases.

Between the two steady state strategies, the *Random Parent Replacement* performed better compared to the *Serial Parent Replacement*. Thus, we conclude that the steady state model is useful over the generational model, and in particular *Random Parent Replacement* is a preferred strategy. The *Steady State Updates* allow newly created good solutions to act as parents in the same generation, and thereby speeding the progress towards the optima.

It is worth re-stating that *Best Update* and *Steady State Update* failed to give any improvement with standard DE-without-*Parent-Child Comparison*. However, in presence of *Parent-Child Comparison* a definite performance boost is achieved. These observations indicate the importance of each operation individually and how their interactions synchronize to affect the overall performance.

Table 9. Standard DE + *Best Update* + *Steady State(RPR)*, $F = 0.7$, $CR = 0.95$, $M = 50$.

	F_{elp}			F_{sch}			F_{ros}		
	Best	Median	Worst	Best	Median	Worst	Best	Median	Worst
S_1 with 10^{-1}	**3,800**	**4,350**	**5,150**	**5,500**	**7,150**	**9,600**	**15,050(38)**	**20,050**	**24,800**
S_2 with 10^{-20}	**20,350**	**21,700**	**24,200**	**36,350**	**40,550**	**44,350**	**40,850(38)**	**46,600**	**51,500**

As a natural extension we combine *Best Update* and *Steady State (Random Parent Replacement)* with standard DE. The results are shown in Table 9. The performance of this modified DE turns out to be best so far; indicating that introduced operations act constructively all-together.

```
If (RandomDouble(0.0,1.0)$\leq P_m$)
  For i=1:N
    If (particle.$x_i\leq$=best.$x_i$)
      {
        Low=best.$x_i$ - $\delta$*(best.$x_i$- particle.$x_i$);
        High=best.$x_i$ + $\delta$*(best.$x_i$- particle.$x_i$);
      }
    Else
      {
        Low=best.$x_i$ - $\delta$*(particle.$x_i$-best.$x_i$);
        High=best.$x_i$ + $\delta$*( particle.$x_i$-best.$x_i$);
      }
    End
    particle.$x_i$=RandomDouble(Low, High);
  End
End
```

Fig. 7. Mutation/Perturbation Operator, borrowed from [10], randmoly places a solution around the *best* location.

Table 10. Standard DE + *Best Update* + *Steady State(RPR)* + *Mutation*, $CR = 0.95$, $F = 0.7$, $P_m = 0.25$.

	F_{elp}			F_{sch}			F_{ros}		
	Best	Median	Worst	Best	Median	Worst	Best	Median	Worst
S_1 with 10^{-1}	**2,450**	**3,050**	**3,850**	**5,350**	7,400	**9,300**	11,550(39)	19,450	**24,600**
S_2 with 10^{-20}	**13,200**	**14,700**	**15,700**	37,250	41,700	46,250	43,300(39)	52,900	61,150

Till now we have been successful in improving the performance of standard DE by borrowing ideas from G3-PCX algorithm and modifying the key steps, particularly by including *Steady State(Random Parent Replacement)* and *Best Update*. This emphasizes the fact that a better understanding of *EOSs* at the level of operators can be highly useful in developing and enhancing other *EOSs*.

At this stage, we identify a mutation operator proposed by [10] in context to development of efficient PSO for solving unimodal problems and introduce it as an additional step in the *Generation* sequence. The mutation operator is described in Figure 7. The goal of this mutation operator is to probabilistically (P_m indicating the mutation probability) perturb a newly created child around the *Best* solution. The mutation operator serves following two purposes: (a) It explicitly promotes diversity in the population, and (b) It aids the search around the *Best* region. The motivation of employing this mutation operator arises from the observation that PSO and DE have a similar working principle in a sense that a new solution is created by adding a weighted sum of vectors to a given base solution. Therefore, a mutation operator having worked well for PSO in context to unimodal problems is a natural candidate to be extended for DE. The mutation operator is combined with the best DE so far and results are shown in Table 10. P_m is chosen as 0.25 as done in [10]. The results show a definitive improvement on F_{elp} and a mixed improvement on F_{sch} and F_{ros}. Such trends reconcile with those presented in [10]. The possible explanation for the improved performance on F_{elp} lies in the variable-separable and unimodal properties of this problem.

Table 11. *Elitist-DE*, $CR = 0.95$, $F = 0.7$, $M = 50$.

	F_{elp}			F_{sch}			F_{ros}		
	Best	Median	Worst	Best	Median	Worst	Best	Median	Worst
S_1 with 10^{-1}	4,550	5,100	6,400	5,900	7,650	11,950	24,900 (39)	615,500	800,300
S_2 with 10^{-20}	22,000	24,150	27,950	39,200	46,300	55,050	4.51e-08 *DNC*	1.42e-03 *DNC*	3.99 *DNC*

Figure 8, summarizes several ideas tested on standard DE in this section. The unsuccessful ideas are shown by arrows pointed downwards, whereas arrows upwards show successful modifications. In next section we propose a variant of standard DE, *Elitist DE*, which is based on a generational model and employs elitism.

6.2 Elitist DE

The DE algorithms tested so far have used *Parent-Child Comparison* (*Serial* or *Random*) strategy for elite preservation, and discovered that *Random Parent Replacement* performed better compared to *Serial Parent Replacement*. Even though *Random Parent Replacement* gives the newly created child an opportunity to survive against a randomly chosen individual (even if it is not superior compared to its base parent) it is still too restrictive in terms of child preservation. There seems to be a scope to further modify the *Replacement* plan. To achieve this we introduce a new way to preserve elites; by combining the parent and child populations and then recovering 50% individuals from the combined population as parents for the next generation based on their fitness values. We shall refer to this as *Elitist DE*. Figure 9 depicts population management in $Elitist - DE$. It should be noted that all the components of *Elitist-DE* are same as standard DE except for elite preservation and that it still follows a generational model.

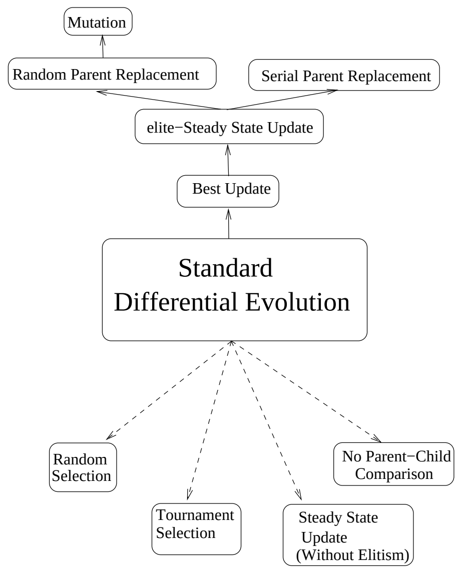

Mutation
Random Parent Replacement
Serial Parent Replacement
elite−Steady State Update
Best Update
Standard Differential Evolution
Random Selection
Tournament Selection
Steady State Update (Without Elitism)
No Parent−Child Comparison

Fig. 8. Elitist DE

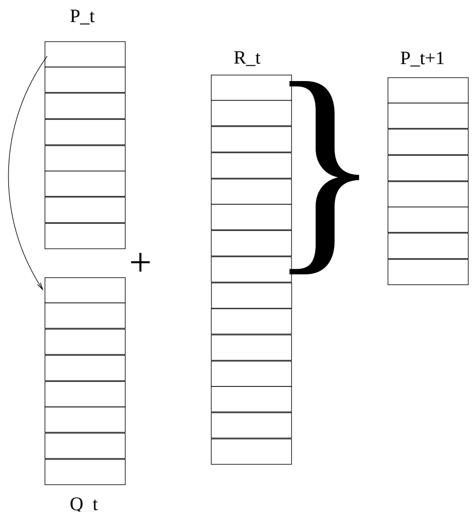


Fig. 9. $Elitist-DE$ based on a generational model. The child population Q_t is created from P_t and a combined pool R_t is formed. From the combined pool 50% of members are retained according to the fitness values.

Table 11 shows that the performance of *Elitist-DE* is superior than standard DE, with respect to both the metrics, on F_{elp} and F_{sch}, and worse on F_{ros}. Since there is a performance enhancement on two test problems we test two previous ideas of *Best Update* and *Mutation* in the *Generation* step, and present results in Tables 12 and 13.

Table 12. $Elitist - DE$ + *Best Update*, $CR = 0.95$, $F = 0.7$, $M = 50$.

	F_{elp}			F_{sch}			F_{ros}		
	Best	Median	Worst	Best	Median	Worst	Best	Median	Worst
S_1 with 10^{-1}	**3,200**	**4,200**	**5,050**	**4,700**	**6,250**	**8,300**	**15,400 (38)**	**19,350**	**23,200**
S_2 with 10^{-20}	**19,250**	**20,850**	**22,000**	**31,000**	**34,550**	**38,850**	**37,550 (38)**	**42,850**	**46,550**

$Elitist - DE$ with *Best Update* performs better in all cases than without *Best Update*, and also shows convergence with S_2 criteria. On comparing $Elitist - DE$ with *Best Update* and standard DE with *Best Update* (Table 7), we find $Elitist - DE$ to be the clear winner in all cases.

The results after including mutation with $Elitist - DE$ along with *Best Update*, for P_m equal to 0.05 and 0.25, are shown in Table 13. On F_{elp}, both the P_m values yield an improved performance. However for F_{sch} and F_{ros}, results with mutation are mixed in nature, but arguably $P_m = 0.05$ yields slightly better performance compared to $P_m = 0.25$.

Table 13. Elitist DE + *Best Update* + *Mutation*, $CR = 0.95$, $F = 0.7$, $M = 50$.

	F_{elp}			F_{sch}			F_{ros}		
	Best	Median	Worst	Best	Median	Worst	Best	Median	Worst
P_m=0.05									
S_1 with 10^{-1}	**2,900**	**3,800**	**4,250**	5,350	**6,100**	8,450	**9,800 (40)**	**19,050**	24,650
S_2 with 10^{-20}	**17,450**	**18,800**	**20,350**	32,450	35,450	39,500	**33,400 (40)**	43,950	49,800
P_m=0.25									
S_1 with 10^{-1}	**2,300**	**2,700**	**3,450**	4,900	6,400	**8,050**	10,650 (37)	19,500	24,900
S_2 with 10^{-20}	**12,500**	**13,550**	**15,250**	32,750	36,350	38,950	39,100 (37)	51,540	57,000

Table 14. Elitist DE + *Best Update* + *Mutation* + Random or Tournament Selection, $CR = 0.95$, $F = 0.7$, $M = 50$, $P_m = 0.05$.

	F_{elp}			F_{sch}			F_{ros}		
	Best	Median	Worst	Best	Median	Worst	Best	Median	Worst
Random Selection of Parents									
S_1 with 10^{-1}	3,200	3,750	4,300	**4,500**	6,300	9,250	9,900(38)	19,550	27,700
S_2 with 10^{-20}	17,400	18,800	20,350	**30,900**	35,550	39,150	34,400(38)	44,250	51,250
Tournament Selection of Parents									
S_1 with 10^{-1}	2,950	3,250	3,950	3,900	5,550	7,050	12,550(33)	18,100	21,800
S_2 with 10^{-20}	15,550	16,300	17,550	**28,750**	**31,800**	**37,050**	34,550(33)	**41,750**	**47,100**

Table 15. G3-PCX and DE's best-so-far performance.

	F_{elp}			F_{sch}			F_{ros}		
	Best	Median	Worst	Best	Median	Worst	Best	Median	Worst
G3-PCX									
S_2 with 10^{-20}	5,744	6,624	7,372	14,643	16,326	17,712	14,847 (38)	22,368	25,797
Best So Far in DE									
S_2 with 10^{-20}	12,500	13,550	15,250	31,000	34,550	38,850	33,400(40)	43,950	49,800

6.3 Last Modifier – Introducing PCX in DE

The overall best performance from all the modified DEs is compared against G3-PCX in Table 15, and the DE performances are unable to match-up with those of G3-PCX. This leads to debate whether standard DE and its variants are inferior compared to G3-PCX, at least on the class of given problems? The EC literature is flooded with studies favoring superiority of one algorithm or paradigm over another. However, authors favor not to side with any particular paradigm, at least at the onset of any task. In our opinion, a modest approach would be to look for features and properties of an algorithm which are needed and make an algorithm effective for a given task. If the algorithm is ineffective then the basic steps should be modified accordingly. Rich EC literature serves as a good source for this purpose in borrowing ideas and operators which have worked successfully in a related context. This is what we attempt next.

While facing a similar predicament with PSO for solving unimodal problems [10] authors successfully introduced a parent-centric *Generation* mechanism based on PCX operator and enhanced the PSOs performance. To achieve this, step *a* of *Generation*, shown in Figure 3, is replaced by PCX operation in which child is created around the current best solution in the population.

6.4 PCX based Generation: Description

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 [9]. The probability distribution is centered around the currently best parent solution. 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 used) 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:

$$\boldsymbol{y} = \mathbf{x}^{(p)} + w_\zeta \|\mathbf{d}^{(p)}\| \boldsymbol{e}^{(p)} + \sum_{i=1,\ i \neq p}^{n} w_\eta \bar{D} \boldsymbol{e}^{(i)}, \tag{4}$$

where $\boldsymbol{e}^{(i)}$ are the $(n-1)$ 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 make the PCX child-creation operation similar in principle to a DE approach, we use the globally best known solution X_{best}, and two randomly chosen parents for any given index parent. The child is created from these three parents, and index parent still acts as the donor parent for the child created. In short, *Generation* plan of the DE proceeds exactly the same except for the fact that now a parent-centric operation is carried out on 3 parents in place of step (a) in Figure 3. The two parameters, σ_ζ and σ_η, required in PCX operation are chosen as 0.1 [9].

6.5 Post PCX Performance

The PCX operation with standard DE (referred to as PCX-DE) failed to give any satisfactory results. Following which we introduced *Best Update* strategy i.e. as soon as a new child was created X_{best} (location around which solutions are being created) was checked for updation. The performance of 'PCX-DE with *Best Update*' was studied and it was found that a population size of $M = 100$ and higher values of CR (taken here as 0.95) yielded an overall better performance. The obtained results were better than the best-so-far DE results. We also tried mutation operator in conjunction with DE-PCX and discovered a degradation in performance. Degradation in performance was explained based on the fact that mutation brings undesirable randomness in the child creation and destroys the ellipsoidal distribution generated from PCX operation.

Next, we introduced the *Steady State(Random Parent Replacement)* in DE-PCX with *Best Update* and observed a slight improvement in few cases, Table 16.

As a next step, the index parent was selected randomly as opposed to being selected serially. Random selection of index parent further improved the performance but still did not take it closer to G3-PCX. It should be noted that at this stage, we have modified all the three steps of standard DE (*Selection*, *Generation* and *Replacement*) by corresponding steps of G3-PCX algorithm. The only difference lies in the child solution retaining 5% traits of the base parent (since CR=0.95). By increasing the value of CR equal to 1.0 i.e. retaining no parent traits, we achieve a performance similar to that of G3-PCX, Table 17. At this point it should be noted that the modified DE and G3-PCX bear algorithmic congruence.

Since components of DE have been modified bit-by-bit till DE becomes same as G3-PCX, a similarity in performance should not be surprising. The path followed to reach this algorithmic similarity can be summarized in two points: (i) Breaking down DE into standard components and studying them under the *Unified Approach* plan, and (ii) Replacing and modifying standard steps by operations which have proven to work successfully elsewhere. The procedure appears to be simple but this paper has made an attempt to amply demonstrate it through several experimentation, which is the main contribution in this paper.

Table 16. Stages in development of DE-PCX

	F_{elp}			F_{sch}			F_{ros}		
	Best	Median	Worst	Best	Median	Worst	Best	Median	Worst
PCX-DE + *Best Update*, $F = 0.7$, $CR = 0.95$, $NP = 100$.									
	Best	Median	Worst	Best	Median	Worst	Best	Median	Worst
S_1 with 10^{-1}	1,800	2,300	3,000	4,500	5,300	6,900	19,600(36)	23,300	27,000
S_2 with 10^{-20}	10,200	11,300	12,100	30,300	33,400	38,600	43,900(36)	50,700	57,800
PCX-DE + *Best Update* + *Steady State (RPR)*, $F = 0.7$, $CR = 0.95$, $NP = 100$.									
S_1 with 10^{-1}	1,800	2,300	**2,900**	**3,900**	**5,200**	**6,600**	21,500(39)	24,500	31,100
S_2 with 10^{-20}	**9,300**	**10,200**	**11,600**	**28,000**	**32,500**	**35,000**	45,500(39)	55,300	66,500
PCX-DE, *Random Parent Selection*, *Best Update*, *Steady State RPR* with $F = 0.7$, $CR = 0.95$, $NP = 100$									
S_1 with 10^{-1}	**1,700**	**2,200**	**2,700**	3,900	5,500	6,700	**16,200(39)**	25,300	30,700
S_2 with 10^{-20}	**8,900**	10,300	**11,400**	28,100	**32,300**	36,300	**42,900(39)**	55,700	67,700

7 Scale-up Performances

Many real-world optimization problems involve large number of variables. So far the unimodal problems with 20 variables were considered. Since the test problems chosen in this paper are scalable, it is interesting to see how do several

Table 17. {PCX-DE + *Random Parent Selection* + *Best Update* + *Steady State (RPR)*, $F = 0.7$, $CR = 1.0$, $NP = 100$} = {G3-PCX }

	F_{elp}			F_{sch}			F_{ros}		
	Best	Median	Worst	Best	Median	Worst	Best	Median	Worst
S_1 with 10^{-1}	**1,000**	**1,400**	**1,900**	**2,300**	**2,800**	**3,300**	**8,800(42)**	**11,700**	**14,400**
S_2 with 10^{-20}	**5,700**	**6,300**	**6,900**	**13,700**	**15,200**	**16,500**	**19,500(42)**	**23,800**	**27,800**

algorithms discussed in this paper perform. For scale-up study we select following four algorithms: standard DE, modified DE (standard DE with *Best Update* and *Steady State(RPR)*), G3-PCX, and a well known evolutionary strategy CMA-ES [15].

Each of the three function is tested for $n = 10$, 20, 50, 100, 150 and 200. For F_{elp}, a relatively simple test problem, $n = 500$ is also tested. For each experiment median and worst number of function evaluations are evaluated over 20 runs. A termination criteria dependent on the problem size (n) is chosen as $\frac{n}{20}10^{-10}$ (i.e. a run is terminated when the best fitness in the population falls below the criteria).

Based on preliminary experimentation, standard and modified DE used following parameter settings: $F = 0.7$, $CR = 0.95$, $NP = 30$. As the number of variables were increased, DE algorithms did not require any scaling of population.

For G3-PCX, an increase in the population size (NP) was required [9]. For F_{elp}, F_{sch} and F_{ros} NP was varied as $3n$, $4n$ and $6n$, respectively. Values of w_ζ and w_η were decreased accordingly. CMA-ES from [15] was employed with default values of population size (NP) and σ.

The median number of function evaluations along with the error bars are plotted in log-log scale, Figures 10(a), 10(b) and 10(c) for F_{elp}, F_{sch} & F_{ros}, respectively. The scale-up plots lead to following observations:

1. The linear trends on the log-log scale reveal that the number of function evaluations rise polynomially with the increase in problem size for all the algorithms (except for piece-wise linear behavior of CMA-ES on F_{elp}) .
2. The number of function evaluations for standard DE are always greater than the modified DE for all the problems, and the gap decreases as the problem size increases. This implies that the *Steady State Update* and *Best Update* become less effective on problems with increasing size.
3. G3-PCX is the best performer for F_{sch} and F_{ros}, however for F_{elp} the performance degrades with increase in problem size. This might attributed simplicity of the F_{elp} problem (where variables have no linkages) and PCX operation is unable to show its effectiveness in a simple case.
4. CMA-ES shows a fluctuating performance, highly efficient on F_{elp} and comparably worst on F_{ros}.

Overall it can be concluded that efficiently designed algorithm (G3-PCX and CMA-ES) have a definite advantage on problems with smaller number of variables. As the problem size increases the behavior of these algorithms becomes unpredictable. The possible reason for unpredictable performance of these algorithms largely lies on the choice of correct parameter settings; which is a challenging task to achieve for long and time consuming simulations. Since the fine-tuning for a given application is a one time investment, it's worth and usefulness is justified.

8 Conclusion and Looking Ahead

Drawing concepts from existing literature, this paper makes a novel attempt in developing a *Unified Approach* towards *Evolutionary Optimization Systems* such as GAs, ES, PSO, DE and EP. An *EOS* was described in four basic steps: *Initialization*, *Selection*, *Generation* and *Replacement*. A four step decomposition of an *EOS* is shown to be useful in understanding the role and modification of each step within the embodiment of an *EOS*. This was demonstrated by evaluating the performance of DE, in context to unimodal problems, compared to a benchmark algorithm G3-PCX. The key steps of DE *Selection*, *Generation* and *Replacement* were modified one-by-one while drawing concepts from G3-PCX. By incorporating features of G3-PCX within the standard DE (such as *Best Update*, *Steady State Update*, mutation, etc.), a certain degree of improvement was obtained at each step. Even after several modifications, DE variants failed to perform as efficiently as G3-PCX. At this stage a final modification in the *Generation* step was made by introducing PCX operation. Along with other alterations and parameter tuning the modified DE was found to be algorithmically equivalent to G3-PCX and showed similar performance. The exercise shows that by employing *Unified Approach* plan two seemingly different algorithms, DE and G3-PCX, can be converted from one into another by modifying the key steps systematically.

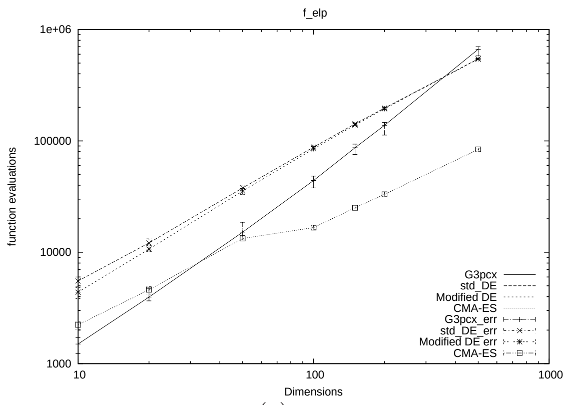
The questions such as What to borrow? and What to modify? cannot be answered unless a systematic plan is put in place. The *Unified Approach* should guide the modification procedure as opposed to a trial and error approach of infinitely many alterations possible in an algorithm. It is expected that illustrations made in this paper should facilitate any such attempt and enable researchers in Evolutionary Computation to adopt unified approach towards evolutionary algorithms and work towards identifying the properties of key steps, useful in order to develop efficient EAs for any given task.

References

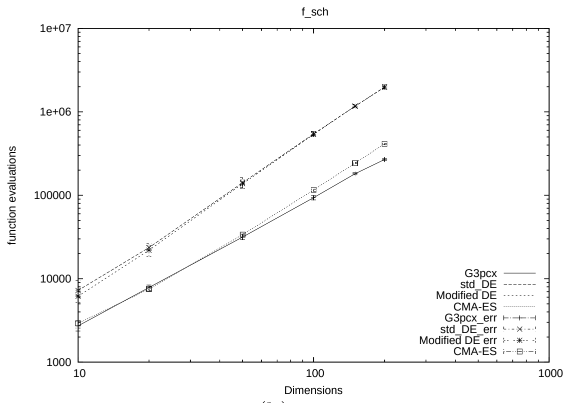
1. H. Abbas. The self-adaptive pareto differential evolution algorithm. In *Proceedings of the 2002 Congress onEvolutionary Computation*, pages 831–836.
2. M. M. Ali and A. Törn. Population set based global optimization algorithms: some modifications and numerical studies. *Computers and Operations Research*, 31:1703–1725, 2004.

3. Hans-Georg Beyer and Department Of Computer Science. Toward a theory of evolution strategies: Self-adaptation, 1995.
4. J. Brest, S. Greiner, B. Boskovic, M. Mernik, and V. Zumer. Self-adapting control parameters in differential evolution: a comparative study on numerical benchmark problems. *IEEE Transactions on Evolutionary Computation*, 6:646–657, 2006.
5. M. Clerc. *Particle Swarm Optimization*. ISTE Ltd, UK/USA, 2006.
6. K. Deb. *A Population-Based Algorithm-Generator for Real-Parameter Optimization*. KanGAL Report Number 2003003.
7. K. Deb. *Multi-objective Optimization Using Evolutionary Algorithms*. John Wiley and Sons, Dordrecht, 2001.
8. K. Deb, S. Agarwal, and T. Meyarvian. A fast and elitist multi-objective genetic algorithm: Nsga-ii. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002.
9. K. Deb, A. Annand, and D. Joshi. A computationally efficient evolutionary algorithm for real-parameter optimization. *Evol. Comput.*, 10(4):371–395, 2002.
10. K. Deb and N. Padhye. Development of efficient particle swarm optimizers by using concepts from evolutionary algorithms. In *Proceedings of the 2010 GECCO conference companion on Genetic and evolutionary computation*, pages 55–62, New York, NY, USA. ACM.
11. D. B. Fogel. An evolutionary approach to the traveling salesman problem. *Biological Cybernetics*, 60, 1988.
12. D. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, New York, 1989.
13. N. Hansen and A. Ostermeier. Adapting arbitrary normal mutation distributions in evolution strategies: The covariance matrix adaptation. pages 312–317. Morgan Kaufmann, 1996.
14. N. Hansen and A. Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation*, 9:159–195, 2001.
15. N. Hansen and A. Ostermeier. Cma-es source code. $http : //www.lri.fr/~hansen/cmaes\_inmatlab.html$, 2009.
16. J. Holland. *Adaption in Natural and Artificial Systems*. University of Michigan Press, MI, 1975.
17. Kenneth A. De Jong. *Evolutionary Computation: A Unified Approach*. MIT Press, 2006.
18. J. Liu and J. Lampinen. A fuzzy adaptive differential evolution algorithm. *Soft computing- A Fusion of Foundations, Methodologies and Applications*, 9(6):448–462, 2005.
19. Kenneth V. Price, Rainer M. Storn, and Jouni A. Lampinen. *Differential Evolution: a practical approach to global optimization*. Springer-Verlag, Hiedelberg, 2005.
20. A. K. Qin, V.L. Huang, and P. N. Suganthan. Differential evolution algorithm with strategy adaptation for globalnumerical optimization. *IEEE Transactions on Evolutionary Computation*, 13(2):398–417, 2009.
21. G. V. Reklaitis, A. Ravindran, and K. M. Ragsdell. *Engineering Optimization Methods and Applications*. Willey, New York, 1983.
22. J. Rönkkönen and J. Lampinen. On using normally distributed mutation step length for the differential evolution algorithm. In *9th Int. Conf. Soft Computing (MENDEL 2003)*, pages 11–18, 2002.
23. H.-P. Schwefel. *Projekt MHD-Staustrahlrohr: Experimentelle optimierung einer zweiphasenduese, TTeil I*. Technical Report 11.034/68, 35, AEG Forschungsinstitut, Berlin.

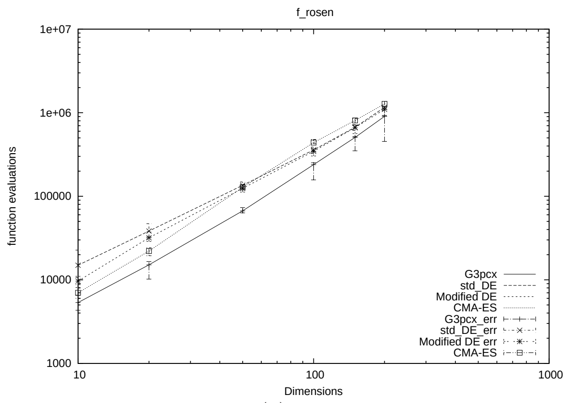
24. Hans-Paul Paul Schwefel. *Evolution and Optimum Seeking: The Sixth Generation*. John Wiley & Sons, Inc., New York, NY, USA, 1993.
25. R. Storn and K. Price. Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces. *J. of Global Optimization*, 11(4):341–359, 1997.



(a)



(b)



(c)

Fig. 10. (a) Scale-up performance for F_{elp} up to 500 variables,(b) Scale-up performance for F_{sch} up to 200 variables, and (c) Scale-up performance for F_{ros} up to 200 variables