Unifying Evolutionary Algorithms and Improving Differential Evolution

Nikhil Padhye
Department of Mechanical Engineering (MechE)
Massachusetts Institute of Technology (MIT) Cambridge, USA
npdhye@mit.edu

Piyush Bhardawaj and Kalyanmoy Deb
Department of Mechanical Engineering
Indian Institute of Technology Kanpur
PIN 208016, Uttar Pradesh, India
{piyush,deb}@iitk.ac.in

July 23, 2010

Abstract

In past, only a few attempts have been made in adopting a unified outlook towards different paradigms in Evolutionary Computation. 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 complex problems. One such attempt is made in this paper, where we reinstate 'Unified Theory Of Evolutionary Computation', drawn from past studies, and investigate four steps – *Initialization*, *Selection*, *Generation* and *Replacement*, which are sufficient to describe common *Evolutionary Optimization Systems* such as Genetic Algorithms, Evolutionary Strategies, Evolutionary Programming, Particle Swarm Optimization and Differential Evolution. As a next step we consider Differential Evolution, a relatively new evolutionary paradigm, and discover its inability to efficiently solve unimodal problems when compared against a benchmark Genetic Algorithm. Targeted towards enhancing DE's performance, several modifications are successfully proposed and validated through simulation results. The *Unified Approach* is found helpful in understanding the role and re-modeling of DE steps to efficiently solve unimodal problems.

1 Introduction

Much of the early research and development in evolutionary based computational methods occurred independently without any interaction(s) among various groups [8]. 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 which gave birth to Differential Evolution (DE) – all now being tagged under Evolutionary Computation and collectively referred to as Evolutionary Algorithms (EAs).

In-spite of advances in different EA paradigms there has been a lukewarm interest in investigating a framework which is capable of explaining the overall behavior of an EA. A plausible

approach could be to decompose an EA into key standard components. Then, by understanding the role of each component individually and interaction between the components, insights into the performance of an EA could be obtained. Attempts in past, adopting *Unified Approach* towards EC have already been made, for e.g., see [8], [1]. This paper samples concepts from these two studies and presents a *Unified Approach* for evolutionary algorithms (GA, ES, EP, PSO and DE) in context to real-parameter optimization for unimodal problems. The primary focus is on the performance of standard DE algorithm on same class of problems, compared against a benchmark genetic algorithm named G3-PCX [3]. After discovering inefficient DE performance, *Unified Approach* is adopted in analyzing major 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 the DE performance is enhanced to an extent where it is comparable to the benchmark results, and the resulting algorithm is found to be equivalent to G3-PCX. Thus, this study highlights that how one can traverse from modifying one algorithm into the other by altering the major steps of an algorithm on the basis of functional requirements, and stresses on the importance of similarities and differences in terms of key steps of an algorithm which give rise to a difference in performance.

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 test problems chosen in this paper and description on experimental methodology. The performance of benchmark algorithm, G3-PCX, specifically designed to solve unimodal problems is also presented. Sections 4 and 5, see performance of standard DE and several modifications on it for the sake of improvement. Details on several proposed strategies for modifications are provided, along with the reasoning for their effectiveness or ineffectiveness. Section 6, concludes the paper and hints on the direction for the future work.

2 Unified Framework For Evolutionary Algorithms

The most notable *breaking new ground* attempt in adopting *A Unified Approach* towards Evolutionary Computation is made in [8], 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 utilized in better understanding (and thereby improving) an EA paradigm? In [8] author outlines a general *Evolutionary Optimization System(EOS)*, which is based on Darwinian evolutionary system, shown in Figure 1.

Figure 1: Evolutionary Optimization System based on *EV-OPT* proposed by [8]

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*.

EOS can be 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 randmomly, (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 *Initialization*, *Selection*, *Generation* and *Replacement* are iteratively repeated till some termination criterion is met. Although detailed descriptions 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.

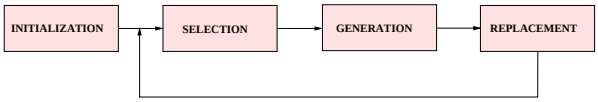

Figure 2: For major Steps in an *EOS*

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. We adopt the notation from [8] while representing evolutionary systems as follows– 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, in this paper, are associated with real parameter optimization and solutions shall be represented as vectors of real parameter decision variables. The initialization of population for *EOS*s shall be done randomly. Next, popular EA paradigms are discusses as instances of *EOS(m,n)*.

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 are performed to create a new population in an iterative manner. Following briefly describe the major steps:*Selection* – Role of this operator is to *prefer better solutions to worse ones*. Individuals are chosen as parents either *deterministically* and *stochastically*. *Generation(Crossover and Mutation)*– Crossover operator creates solution(s) from selected parent(s) and mutation operator modifies the created offspring randomly. *Replacement* – can be carried out based on either steady state or generational model, though it is worthwhile to mention that standard genetic algorithms ([7], [6]) utilized a generational model in which parents survived for exactly one generation and completely replaced by their offsprings. Thus, standard GAs could be thought of as EOS(m,m).

2.2 Evolutionary Strategies as EOS(m,n)

Evolutionary Strategies (ES) [10] have been fundamentally different from binary GAs principally in two ways: (1) ESs used real parameter values, and (2) early ESs did not have any crossover-like operators. Popular $(\mu + \lambda)$-ES can be represented in form of *EOS*(μ, λ) as follows: *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 and

Recombanative Evolutionary Strategies $(\mu/\rho + \lambda)$-ES and $(\mu/\rho, \lambda)$-ES can also be represented in form of *EOS(m,n)* by appropriately modifying the key steps.

2.3 Evolutionary Programing as EOS(m,m)

Evolutionary Programming (EP) is a mutation based evolutionary algorithm applicable to real-parameter optimization [5]. EP is similar to ES in the fact that normally distributed mutations are performed in both algorithms. In standard EP, each population member is deterministically selected to create offspring. 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.

2.4 Particle Swarm Optimization as EOS(m,m)

In an interesting study done in [4], authors presented an archive-based evolutionary algorithm which was equivalent to Particle Swarm Optimization(PSO). Using their 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).

2.5 Differential Evolution as EOS(m,m)

Differential Evolution (DE) algorithm has emerged as a very competitive form of evolutionary computing more than a decade ago. 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. As majority of simulations presented in this study are based either on standard DE, its pseudo code is presented in Figure 3.

Selection, *Generation* and *Replacement* steps in DE here are same as those in "DE/best/1/exp" [9]. The population is scanned serially and for creation of a child, corresponding to any individual, four parents are selected (i.e. the individual itself, also referred to as base or index parent, *best fitness* individual from the previous generation and any two population members chosen at random). 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 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 this DE should be noted: (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.

Figure 3: Standard Differential Evolution Algorithm *(DE/best/1/exp)*, borrowed from [9].

Input DE parameters: scale factor *(F)*, crossover-rate *(Cr)* and population size *(M)*
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:*
$V_{i,t} = X_{best,t-1} + F \cdot (X_{r1,t} - X_{r2,t})$
(b)*Create* i^{th} *Trial Vector:*
$U_{i,t}$=CombineElements($X_{i,t}, V_{i,t}$) // with probability CR
Replacement
If (Fitness($U_{i,t}$)≤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*.

3 Test Suite

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 [3]:

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

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

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

In all these problems we use $n = 20$ 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 $F^* = 0$. We initialize the population away from the known optima while restricting $x_i \in [-10, -5]$ for all i, in all problems. In subsequent generations we do not confine solutions to lie in the above range. After initialization, 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 choose

a 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 a value of 10^{-20} for this purpose.

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. G3-PCX outperformed other state-of-the-art algorithms [3] and is treated as a benchmark for this study.

Table 1: G3-PCX results, as reported in [3].

	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

4 Performance Analysis of Standard DE

DE algorithm presented in section 2.5 belongs to the DE family of Storn and Price [9]. The family comprises of 10 different *Generation* strategies. Based on preliminary experiments we found *strategy* 1 to yield overall best performance. According to this strategy new solutions are created around the previous generation's *best solution* (step *a* in *Generation*, Figure 3). This feature of generating solutions around the best population member is desirable in solving unimodal problems and also done in G3-PCX [3], hence it is no surprise that *strategy* 1 was the best performer. In remainder of this paper, DE with strategy 1 is employed for simulations and referred to as standard DE. We performed a parametric study on standard DE for M, CR and F, and found $M = 50$, $CR = 0.95$ and $F = 0.7$, as optimal values with respect to all the three test problems. The results of standard DE are reported in Table 2.

5 Functional Analysis and Modification of Standard DE

One of the noticeable features of standard DE is *elitism* at the individual level i.e. a child is compared with its *base parent* (i.e. the individual at the index corresponding to which child has been created), and only the better of the two survives for the next generation. We modified this *Replacement* scheme by always accepting the newly created child i.e. without carrying out the parent-child comparison. This resulted in a significant performance degradation in all three test problems with respect to both the metrics, indicating that elitism in DE by parent-child comparison is key to its performance.

Next, we try two *Selection* schemes, *Tournament* and *Random*, instead of the usual serial parent selection. The results are shown in Table 3 indicate that the alternate selection schemes perform poorly compared to serial selection, and we conclude that *deterministic* serial approach works most appropriately for DE.

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 problems exibhiting unimodality, and the benchmark algorithm G3-PCX successfully exploits this property.

Table 2: Standard DE, "DE/best/1/exp" [9], $F = 0.7$, $CR = 0.95$, $M = 50$.

	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 3: 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 4: 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

A major difference between G3-PCX and the standard DE arises from the fact that former uses the *current best* location in the population, whereas DE utilizes the *previous generation's best*. We 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 child creation. The results shown in Table 4 reflect an improved performance in all cases. Thus, we conclude that creating solutions around X_{best} is an effective strategy for standard DE while solving unimodal problems.

We take the next step and observe a basic difference in steady state and generational models of G3-PCX and standard DE. In standard DE a newly created child has to wait till next generation before it can be selected as the index parent. We test the steady state version of standard DE in which as soon as a child is created it is compared with its index parent. The index parent is replaced if the child is better. Since the created child is compared with the index parent itself, we refer this as *Serial Parent Replacement*. In another steady state version of standard DE, we compare the created child with a randomly selected member of the population and carry out the replacement. In this version even if the newly created child is inferior to the index parent, it has a chance of surviving while being compared against a randomly chosen individual. The results for both the steady state versions are shown in Tables 5.

Both the steady state versions show an improvement over standard DE (Table 2). The steady state versions are also an improvement over the DE with *Best Update* except for the F_{ros}. Between the two versions, the *Random Parent Replacement* performs 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 more preferred strategy.

Now, we combine *Best Update* and *Steady State (Random Parent Replacement)* with standard DE, results shown in Table 6. The performance of this modified DE turns out to be best so far.

Table 5: Standard DE + $SteadyStateVersions$, $CR = 0.95$, $F = 0.7$, $M = 50$.

	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

Table 6: 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

Table 7: 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, particularly *Steady State* and *Best Update*, from G3-PCX algorithm. 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 also identify a mutation operator proposed by [4] in context to development of efficient PSO for solving unimodal problems. In short, the goal of this mutation operator is to probabilistically (P_m indicating the mutation probability) perturb a newly created child randomly around the *Best* solution. This serves for following two purposes: (a) to explicitly promote the diversity in the population, and (b) aid search around the *Best* region. We combine the mutation operator with the best DE so far (Table 7) and show the results in Table 7. P_m is chosen as 0.25 as done in [4]. 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 [4]. The possible explanation for the improved performance on F_{elp} lies in the variable-separable and unimodal properties of this problem.

5.1 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 child preservation. Here, we introduce a new way to preserve elite by combining the parent and child population and then recovering 50% individuals from the population as parents for the next generation based on their fitness values. This style of preserving elite solutions is adopted in NSGA-II (a widely popular muti-objective genetic algorithm) [2]. We shall refer to this as *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 follows the generational model. Table 8 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*, as present results in Tables 9 and 10. 'Elitist DE with *Best*

Table 8: *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*

Table 9: 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

Table 10: 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

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 2), we find Elitist DE to be the clear winner in all cases. As a final step we also include mutation in 'Elitist DE with *Best Update*' and show results for different values of P_m, 0.05 and 0.25, Table 10. On F_{elp}, mutation with both the P_m values yield an improved performance. For F_{sch} and F_{ros} results with mutation are mixed in nature. However, a better performance on these two functions is obtained with $P_m = 0.05$ than compared to $P_m = 0.25$.

5.2 PCX Based DE

The overall best performance from all the modified DEs is compared against G3-PCX in Table 11, and the DE performances are unable to match-up with those of G3-PCX. While facing a similar predicament with PSO, in [4], authors successfully introduced a parent-centric *Generation* mechanism based on PCX operator and enhanced the PSOs performance, which we attempt next. The step *a* of *Generation*, shown in Figure 3, is replaced by PCX operation in which child is created around the best solution. More details on PCX operator can be found in [3]. Two parameters required in PCX, σ_ζ and σ_η, were taken as 0.1.

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 results were better than best-so-far DE results. We also tried mutation operator in conjunction with DE-PCX and discovered a degradation in performance. This could be explained based on the fact that mutation brings undesirable randomness into the child creation and destroys the ellipsoidal distribution from PCX operation.

Next, we introduce the *Steady State with Random Parent Replacement* in DE-PCX with *Best Update* and observe a slight improvement in few cases, Table 12. As a next step, the index

Table 11: 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

Table 12: PCX-DE + *Best Update* + *Steady State (RPR)*, $F = 0.7$, $CR = 0.95$, $NP = 100$.

	F_{elp}			F_{sch}			F_{ros}		
	Best	Median	Worst	Best	Median	Worst	Best	Median	Worst
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

Table 13: PCX-DE + *Random Parent Selection* + *Best Update* + *Steady State (RPR)*, $F = 0.7$, $CR = 1.0$, $NP = 100$.

	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

parent was selected randomly as opposed to being being selected serially. Random selection of index parent further improved the performance but still did not take it closer to G3-PCX. At this point we increased the value of CR to 1.0 and achieved a performance similar to that of G3-PCX, Table 13.

6 Conclusion

Drawing concepts from existing literature, this paper makes a novel attempt in developing a unified approach towards *Evolutionary Optimization Systems*. The key steps required for describing an *EOS* are *Initialization*, *Selection*, *Generation* and *Replacement*. The central focus of this study is then to improve the performance of standard DE on the class of unimodal problems by identifying modifying its key steps– *Selection*, *Generation* and *Replacement*. Drawing principles from G3-PCX, a benchmark algorithm, key steps in DE are modified one-by-one. At each stage certain degree of performance improvement is obtained. Finally, PCX operation is introduced in standard DE along with the other alterations, and the performance is comparable to G3-PCX. Although, the modified DE is algorithmically equivalent to G3-PCX, the study suggests how two seemingly different algorithms can be converted from one into another by modifying the key steps. Such a study should 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] K. Deb. *A Population-Based Algorithm-Generator for Real-Parameter Optimization.* KanGAL Report Number 2003003.

[2] 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.

[3] K. Deb, A. Annand, and D. Joshi. A computationally efficient evolutionary algorithm for real-parameter optimization. *Evol. Comput.*, 10(4):371–395, 2002.

[4] 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.

[5] D. B. Fogel. An evolutionary approach to the traveling salesman problem. *Biological Cybernetics*, 60, 1988.

[6] D. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley, New York, 1989.

[7] J. Holland. *Adaption in Natural and Artificial Systems.* University of Michigan Press, MI, 1975.

[8] Kenneth A. De Jong. *Evolutionary Computation: A Unified Approach.* MIT Press, 2006.

[9] Kenneth V. Price, Rainer M. Storn, and Jouni A. Lampinen. *Differential Evolution: a practical approach to global optimization.* Springer-Verlag, Hiedelberg, 2005.

[10] H.-P. Schwefel. *Projekt MHD-Staustrahlrohr: Experimentelle optimierung einer zweiphasenduese, TTeil I.* Technical Report 11.034/68, 35, AEG Forschungsinstitut, Berlin.