Soft Comput (2004)
DOI 10.1007/s00500-004-0377-4

FOCUS

K. Deb

A population-based algorithm-generator for real-parameter optimization

Published online: 7 June 2004

Abstract In this paper, we propose a population-based, four-step, real-parameter optimization algorithm-generator. The approach divides the task of reaching near the optimum solution into four independent plans of (i) selecting good solutions from a solution base, (ii) generating new solutions using the selected solutions, (iii) choosing inferior or spurious solutions for replacement, and (iv) updating the solution base with good new or old solutions. Interestingly, many classical and evolutionary optimization algorithms are found to be representable by this algorithm-generator. The paper also recommends an efficient optimization algorithm with the possibility of using a number of different recombination plans and parameter values. With a systematic parametric study, the paper finally recommends a real-parameter optimization algorithm which outperforms a number of existing classical and evolutionary algorithms. To extend this study, the proposed algorithm-generator can be utilized to develop new and more efficient population-based optimization algorithms. The treatment of population-based classical and evolutionary optimization algorithms identically through the proposed algorithm-generator is the main hall-mark of this paper and should enable researchers from both classical and evolutionary fields to understand each other's methods better and interact in a more coherent manner.

Keywords Real-parameter optimization · Evolutionary optimization · Multi-point search · Algorithm-generator

1 Introduction

Over the years, many real-parameter optimization algorithms have been developed by using point-by-point (Rao 1984; Reklaitis et al. 1983) as well as multi-point approaches (Goldberg 1989; Holland 1975; Fogel 1995; Schwefel 1995). While a point-by-point approach begins with one guess solution and updates the solution iteratively in the hope of reaching near the optimum solution, a multi-point approach deals with a number of solutions in each iteration. Starting with a number of guess solutions, the multi-point algorithm updates one or more solutions in a synergistic manner in the hope of steering the population towards the optimum. In this paper, we focus our attention to the multi-point optimization algorithms, which we shall refer here as the *population-based* optimization algorithms. Although there exists a few classical population-based optimization algorithms, almost all evolutionary optimization algorithms are population-based algorithms.

In the remainder of this paper, we suggest a generic population-based algorithm-generator for real-parameter optimization. The main advantage of the proposed four-step algorithm-generator is the *functional decomposition* of four important tasks needed in real-parameter optimization. The tasks (we refer them as *plans*) are independent to each other, thereby enabling an user to study the effect of each plan on the complete algorithm and should make it easier to develop efficient population-based optimization algorithms. We then demonstrate how the proposed algorithm-generator can be used to represent a number of classical and evolutionary algorithms in a functionally decomposed manner.

One of the important tasks in an optimization algorithm is the generation of new solutions from existing solutions. In this paper, we review a number of commonly-used evolutionary generational plans and suggest an efficient operator for this purpose. With a recently suggested efficient optimization algorithm (the generalized generation gap (G3) model (Deb et al. 2002)) and an efficient parent-centric recombination operator used as a generational plan, we then perform a detail study to identify optimal parameter values for the G3 model. Finally, we compare the performance of the resulting well-parameterized optimization algorithm (G3 model) with a

K. Deb
Kanpur Genetic Algorithms Laboratory (KanGAL),
Indian Institute of Technology Kanpur,
Kanpur, 208 016, India
E-mail: deb@iitk.ac.in, http://www.iitk.ac.in/kangal/pub.htm

number of existing classical and evolutionary algorithms. In almost all cases, the G3 model with a couple of versions of the parent-centric recombination operator evolved as the winner. Besides the suggestion of a successful algorithm here, the other main contribution of this paper is the proposed algorithm-generator, which is generic and can be used to systematically develop other more efficient population-based optimization algorithms. The proceedings of this paper also demonstrates a case study of parameterizing and evaluating such an algorithm.

2 Population-based optimization algorithms

Optimization algorithms can be classified into two main categories: (i) gradient-based methods, in which first-order (and sometimes second-order) derivative vectors are used to drive the search process and (ii) direct search methods, in which only objective function values are used to drive the search. Although gradients can be computed numerically or automatically, each has their own merits and demerits (Rao 1984; Reklaitis et al. 1983). Since a numerical gradient computation involves more than one function evaluations in the vicinity of a solution, most gradient-based techniques use a point-by-point approach, in which one solution (or point) is updated to a new and hopefully better solution. Without the use of any gradient information, most direct search methods explicitly employ more than one solutions in each iteration. The presence of multiple solutions is then exploited to find better search regions. It is these multi-point approaches, which are of interest to this paper. We call these algorithms population-based approaches, as they require a population of solutions in one iteration, instead of just one solution. Some of the population-based classical optimization algorithms are Hooke-Jeeves pattern search method (Reklaitis et al. 1983), Nelder and Meade's simple search method (Nelder and Mead 1965), Box's complex search method (Box 1965) and a plethora of adaptive random search methods discussed in texts (Deb 1995; Reklaitis et al. 1983; Rao 1984). Among the non-traditional optimization methods, evolutionary search algorithms (involving genetic algorithms (GAs), evolution strategies (ESs), evolutionary programming (EP), differential evolution (DE)) are direct population-based approaches (Holland 1975; Goldberg 1989; Rechenberg 1965; Storn and Price 1997; Fogel et al. 1995).

In the following, we outline some of the advantages of using a population-based optimization algorithm:

1. In principle, gradient-like information can be implicitly obtained by explicitly processing multiple solutions. This would be beneficial in problems in which gradient computations may be difficult or erroneous with a numerical scheme.
2. A population of solution provides the information of a good search *region*, instead of only a good point, in the search space.
3. In addition to representing a potentially good search region, the variance in the population members provide a good information about the extent of the potential search region. If new solutions are created in proportion to the variance of existing population of points, a *self-adaptive* search procedure can be developed. In the start of such an algorithm with a randomly-picked initial population of solutions, the variance in the population members is expected to be large, thereby ensuring a thorough search of the entire space. On the other hand, during later iterations when the population of points have converged near the optimum, the variance in population members is expected to be small, thereby ensuring a focused search near the optimum. Without an external guidance, such a population-based algorithm can widen or narrow down its search power adaptively.
4. Each iteration of such a population-based algorithm can use a different form of the underlying objective function. Since a comparison scheme would usually be followed to establish a hierarchy of population members, a rough estimate of the quality of the solutions is enough to drive the search. For highly computationally intensive objective function evaluation procedures and for problems where a clear mathematical or procedural objective function evaluation is not possible (such as the aesthetics of a design being an objective to be maximized), such a comparative evaluation procedure is certainly beneficial. It is not clear how a similar task can be achieved with a point-by-point optimization approach, in which there exist only one solution in each iteration.
5. There always lies safety in numbers. The presence of multiple solutions in a search process allows diversity to be maintained. Although this can cause a computational overhead in simpler problem-solving tasks, they can become useful in avoiding sudden local convergence behavior, common to many point-by-point optimization algorithms. Even in harder problems, the processing of multiple solutions in one iteration may constitute a *parallel* search, by sharing the discovered good sub-solutions among different population members.
6. The presence of multiple solutions can be beneficial in handling constraint-optimization problems. Although some point-by-point approaches can only be started with a feasible solution, most population-based approaches can work with some feasible and some infeasible solutions and some approaches can work with infeasible solutions alone. A careful hierarchy of infeasible and feasible solutions can be maintained in a population to steer the search towards potential feasible regions. One such population-based constraint-handling approach is developed elsewhere (Deb 2001), which does not require any penalty parameter while establishing the hierarchy among feasible and infeasible solutions.

7. Finally, there exist a number of other optimization tasks, such as multi-modal and multi-objective optimizations, in which the task is to find multiple optimal solutions. For handling these problems, a point-by-point approach has no other option but to apply the algorithm again and again to find multiple different solutions. On the other hand, a population-based approach can be adapted to find and capture multiple optimal solutions simultaneously in a single simulation run (Deb 2001; Goldberg and Richardson 1987). It is intuitive to realize that the simultaneous discovery of multiple optimal solutions is possible to achieve by an implicit parallel approach within a population-based algorithm, whereas in multiple applications of one algorithm, any property common to the multiple optimal solutions must have to be rediscovered in every application independently, a matter which is well addressed in a recent study (Deb 2003).

In the next section, we suggest a population-based algorithm-generator for real-parameter optimization.

3 A population-based algorithm-generator for optimization

As the name suggests, a population-based optimization algorithm begins its search with a population of guess solutions. Thereafter, in each iteration the population is updated by using a population-update algorithm. We assume that the algorithm at iteration t has a set of points $B^{(t)}$ (with $N = |B^{(t)}|$). At the end of each iteration, this set is updated to a new set $B^{(t+1)}$ by using four user-defined plans. For brevity, here we drop the superscript denoting the iteration counter from the sets.

Population-update-algorithm(SP, GP, RP, UP)

Step 1 Choose μ solutions (the set P) from B using a *selection plan* (SP).
Step 2 Create λ solutions (the set C) from P using a *generation plan* (GP).
Step 3 Choose r solutions (the set R) from B for replacement using a *replacement plan* (RP).
Step 4 Update these r members by r solutions chosen from a comparison set of R, P, and C using a *update plan* (UP).

Since by choosing a plan for each step one develops a new optimization algorithm, we call the above procedure an algorithm-generator. Figure 1 shows a sketch of the proposed generic procedure. The first step chooses μ solutions from the solution bank B (all solutions in the top-left plot) for their use in creating new solutions in the second step. Thus, the selection plan (SP) for choosing these μ solutions must emphasize the *better* solutions of B. The chosen solutions ($\mu = 3$ in the figure) are shown as dark-outlined points on the top-left plot and are also joined by lines. Clearly, there are two ways to choose a good SP. A set of μ solutions can be chosen either by directly emphasizing the better solutions in B or by de-emphasizing the worse solutions of B. The algorithm can be made somewhat 'greedy' by always including a copy of the 'best' solution of B in P.

In the second step, λ new solutions are created from the chosen set P by using a generation plan GP. One convenient procedure would be to choose a probability distribution function (PDF) using the set P and create λ solutions. Figure 1 shows a typical probability distribution in which solutions around the members of P are emphasized. Members of the set C ($\lambda = 4$ in the figure) are shown in solid circles in the figure. This is the most crucial step of the proposed algorithm-generator. We review some existing generational plans and suggest a new one in Sect.5. Although for generating each new solution a different PDF can be used thereby requiring

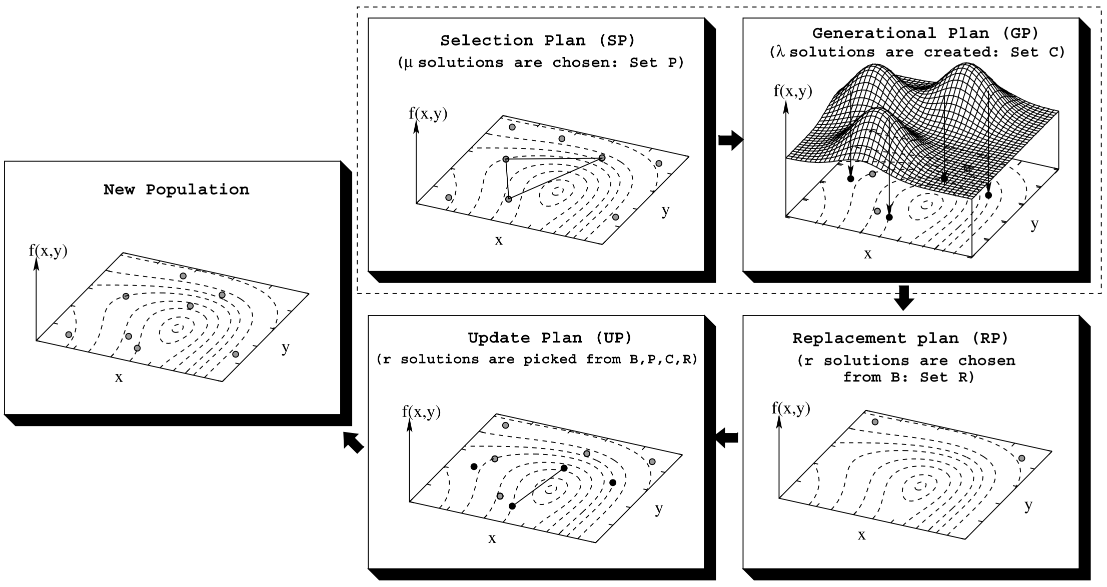
Selection Plan (SP)
(μ solutions are chosen: Set P)
Generational Plan (GP)
(λ solutions are created: Set C)
New Population
Update Plan (UP)
(r solutions are picked from B,P,C,R)
Replacement plan (RP)
(r solutions are chosen from B: Set R)
f(x,y)
x
y

Fig. 1 A sketch of one iteration of the proposed population-based optimization procedure. For some realizations, the SP and GP steps can be applied multiple times before proceeding to the RP and UP plans

the need to use SP-GP steps many times before proceedings further, for simplicity one PDF can be chosen to create all λ new solutions.

In the third step, r solutions are chosen from the solution bank B for a replacement using the newly created solutions. Here, different replacement plans are possible. The RP can simply choose r solutions at random or include some or all members of P to ensure diversity preservation. The RP can also pick a set of bad solutions from B to constitute a faster search. In the figure, r ($= 2$ in the figure) worst solutions from B are chosen as members of R.

In the fourth step, the r chosen members are updated by r members chosen from R, P, and C by using an update plan (UP). It is intuitive that the UP must emphasize the better solutions of $R \cup P \cup C$ in this step. However, the r slots can also be updated directly from members of C alone or from a combined population $R \cup C$ or $P \cup C$. In the former case, the population update algorithm does not have the elite-preserving property. To really ensure elite-preservation, the RP should choose the best solutions of B and a combined P and C set needs to be considered in Step 4. In the figure, the best r solutions (joined by a line) from the $R \cup P \cup C$ are chosen and the originally chosen r solutions in Step 3 are replaced by these r solutions.

Although the above decomposition principle is not the only possible way to describe an optimization algorithm-generator, it certainly makes a *functional decomposition* of the salient tasks needed in a good search and optimization algorithm. The selection plan emphasizes better solutions from a population, the generational plan uses these solutions to explore the search region around them and create new solutions, the replacement plan picks potentially bad solutions from the original population to be replaced, and finally the update plan completes the replacement task with the newly created solutions. Each plan controls a crucial property of the resulting algorithm. By properly choosing a strategy for each plan, the resulting optimization algorithm can be made *greedy* to mainly solve unimodal problems, can have a global perspective for targeting to find the global optimum, or can be made to be a monotonically non-deteriorating algorithm to have sustained improvement in the solution quality.

The procedure of using the above algorithm-generator for developing a new optimization algorithm is as follows:

1. Choose a suitable plan for each step (SP, GP, RP, and UP).
2. Perform a parametric study to discover optimal algorithm parameters, such as N, μ, λ and r.

Of course, in order to develop an efficient optimization algorithm, the first task above must have to be optimized somewhat. A few potential candidate plans for each of the four steps can be first chosen and then compared by applying each of the resulting algorithms to a set of carefully chosen test problems. Each such algorithm must have to be studied with its optimal parameter setting. We shall demonstrate the proceedings of one such study later in this paper. But, first we review a number of population-based classical algorithms and discuss how the above algorithm-generator can be used to describe each of them in a unified manner. Thereafter, we shall discuss the same for a number of real-parameter evolutionary algorithms.

4 Classical methods

Although most classical optimization methods use a point-by-point approach, in which only one solution is used and updated in each iteration, there exist a number of population-based techniques. Without loss of generality, we discuss here a few algorithms used for minimizing objective functions.

4.1 Evolutionary optimization

Box's (1957) evolutionary optimization technique (not to be confused with the evolutionary algorithms described in the next section) used a structured set of solutions in each iteration. For n decision variables, the set B contains $N = 2^n + 1$ solutions, one having at each corner of an orthogonal hyper-box formed using the coordinate directions. Figure 2 shows a set of $(2^2 + 1)$ or 5 points for a $n = 2$-variable optimization problem. Based on the objective function values at all these corner points, a new hyper-box (hence a new set of solutions) is formed around the best solution (having the smallest objective function value). In the context of the proposed algorithm-generator, this method uses a selection plan (SP) in which only the best solution is chosen from B, thereby making $\mu = |P| = 1$. The generational plan (GP) then creates $\lambda = 2^n$ more solutions (the set C) systematically around this best solution. Figure 2 illustrates this procedure. In the third step, all members of B are chosen as R and in the fourth step they are replaced by $P \cup C$. Here are the four plans which constitute Box's evolu-

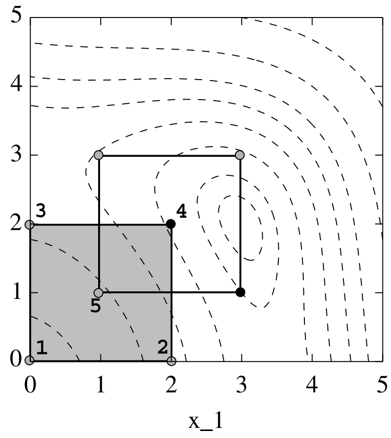

Fig. 2 Box's evolutionary optimization procedure. Solutions 1 to 5 are five members of the initial population

tionary optimization algorithm in terms of our proposed algorithm-generator:

SP: Choose the best of $N = 2^n + 1$ solutions and call it as P. Thus, $\mu = |P| = 1$.
GP: Create $\lambda = 2^n$ solutions (the set C) systematically around the sole member of P (Box 1957; Deb 1995). Thus, $|C| = 2^n$.
RP: Set $R = B$.
UP: Set $B := P \cup C$, ensuring $|B| = N$.

4.2 Simplex search method

Nelder and Meade's simplex search method (Nelder and Mead 1965) uses $N = n + 1$ solutions in the population B. Figure 3 shows a set of three points in the population (simplex) on a $n = 2$-variable problem. The worst simplex member is reflected around the centroid of the rest of the n solutions. Thereafter, depending on the function value comparison of this reflected solution with the the best, next-best, and the worst solution in the population, a new solution is created. The new solution then replaces the worst solution. A few iterations of this simplex search method are illustrated in Fig. 3. The shaded region represent the first simplex. In terms of our algorithm-generator, the four plans for this algorithm are as follows:
SP: Set $P = B$ (that is, all $N = n + 1$ population members of B are chosen as members of P).

GP: Create a reflected solution using members of P. Based on objective function value comparisons of members of P, a new solution C is created by reflection, expansion or contraction, as the case may be (see (Deb 1995; Nelder and Mead 1965) for details). Thus, $|C| = 1$.
RP: Pick the worst solution of B and include it in R. Thus, $|R| = 1$.
UP: Set $B := (B \cup C) \setminus R$.

The complex search method (Box 1965 ;Deb 1995) has a similar search principle to the above and can also be written systematically as above.

4.3 Adaptive random search methods

A number of early studies (Brooks 1958; Luus and Jaakola 1973; Price 1983), introduced and encouraged the use of random search methods in practice, particularly to give a search algorithm its global perspective. In a particular adaptive search method (Luus and Jaakola 1973), a set of N randomly chosen solutions is initialized and the best solution is chosen. Thereafter, a set of N new solutions are created in a hyper-box which is smaller in size compared to the previous hyper-box. Figure 4 shows $N = 6$ solutions in two consecutive iterations. The solutions marked in circles are created at random on the shaded region in the first iteration. The best solution in each iteration is marked with a solid symbol. The figure shows how the next-iteration hyper-box gets reduced in size and placed centered around the best solution. In terms of our algorithm-generator, the four plans are as follows:

SP: Set P with the best solution of B. Thus, $\mu = |P| = 1$.
GP: Create $\lambda = N$ new random solutions C from a hyper-box centered around the sole member of P (see Deb 1995; Luus and Jaakola 1973) for details).
RP: Set $R = B$.
UP: Set $B = C$.

In the last step, all members of B are replaced with the newly created solutions (C).

There exists other complicated adaptive random search methods which use more sophisticated update rules. In a recent approach (the so-called shuffled complex evolution (SCE)) (Duan et al. 1993), a population-based combined simplex and random search method is suggested. The study reported mixed results on a number of test problems. By carefully understanding the tasks needed in each step of the proposed algorithm-generator and by using the salient properties of classical and non-classical optimization methods each step can be designed better. Such an endeavor should allow development of more interesting and efficient optimization algorithms systematically.

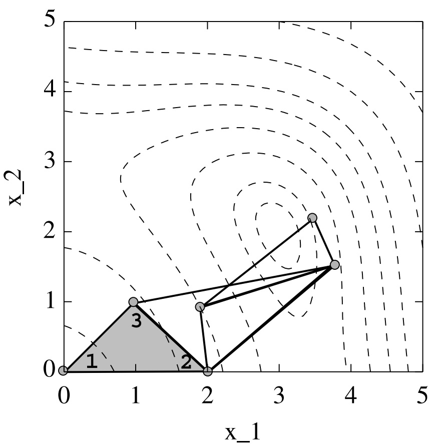

Fig. 3 Nelder and Meade's simplex search method is illustrated. Solutions 1 to 3 are three members of the initial population

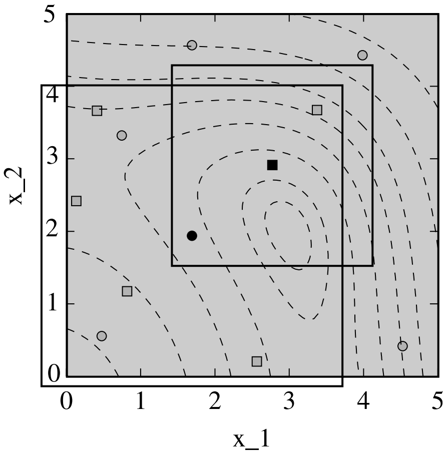

Fig. 4 Adaptive random search method is illustrated. Circles represent $B^{(0)}$ and squares represent $B^{(1)}$

5 Evolutionary methods

Evolutionary algorithms (EAs) are stochastic search algorithms which use randomized search operators (Goldberg 1989; Holland 1975). For real-parameter optimization, different types of evolutionary algorithms are suggested. Of them, the real-parameter genetic algorithms, evolution strategies, differential evolution and evolutionary programming are most popularly used. Before we discuss how some of these algorithms can be represented by the proposed algorithm-generator, we discuss an important matter related to EAs and the algorithm-generator.

To those familiar with EAs, the four steps of the proposed algorithm-generator may seem to be quite motivated from the working principles of an evolutionary algorithm. This is somewhat true since the majority of population-based optimization algorithms in use today follow evolutionary principles. However, the explicit use of the last two steps (RP and UP) are purposeful. To an evolutionary algorithm and to any population-based optimization algorithm, there lies an important issue – the balance between the *exploitation* and *exploration* (Goldberg 1989). The first term is related to the extent of importance given to already discovered solutions in the subsequent iterations and the second term is related to the extent of generalization used to create a new solution from the already available solutions. Studies in the past (Goldberg et al. 1993; Thierens and Goldberg 1993) have shown a way to quantify these two terms in the context of a GA and have clearly shown the need to make a balance between them. We emphasize here that the exploitation aspect of an EA not only depends on the selection plan, but also on the other two plans (RP and UP), a matter which is often ignored in the EA studies. The emphasis of the current-best solutions in a finite-sized population not only depends on which solutions and how often they are selected in the selection plan for creating the mating pool, but also indirectly depends on (i) how many and which solutions are chosen to be deleted from the population and (ii) whether or not the existing population members are compared with the newly created solutions for their further inclusion in the population (this is often used to introduce elite-preservation). The above two tasks are explicitly included in the algorithm-generator as the replacement plan (RP) and the update plan (UP), respectively, to make the algorithm development task flexible. However, the exploration issue can be solely derived from the operations of the GP.

We are now ready to present a number of EAs in the light of our proposed algorithm-generator.

5.1 Real-parameter genetic algorithms

In a real-parameter genetic algorithm (rGA), a solution is directly represented as a vector of real-parameter decision variables. Starting with a population of such solutions (usually randomly created), a set of genetic operations (such as selection, recombination, mutation, and elite-preservation) are performed to create a new population in an iterative manner. Clearly, the selection plan (SP) must implement the selection operator of rGAs, the generational plan (GP) must implement all variation operators (such as recombination and mutation operators), and the update plan (UP) must implement any elite-preservation operator. Although most rGAs differ from each other mainly in terms of their recombination and mutation operators (some commonly-used recombination operators are discussed in the Appendix and are also can be found in the literature), they mostly follow one of a few algorithmic models. We shall discuss some of these models a little later, but before that we shall discuss some commonly-used EA operators which can be explained easily with the help of plans outlined in the proposed algorithm-generator.

5.1.1 Elite preservation operator

The elite preservation in an EA is an important task (Rudolph 1994). This is enforced by allowing best of parent and offspring populations to be propagated in two consecutive iterations. The update plan of the proposed algorithm-generator achieves elite preservation in a simple way. As long as the previous population B or the chosen parent population P is included in the update plan for choosing better solutions, elite preservation is guaranteed. Some EAs achieve this by choosing the best μ solutions from a combined population of P and C.

5.1.2 Commonly-used EA selection operators

Among the EA selection operators, the proportionate, ranking, and tournament selection operators are often used. With the proposed algorithm-generator, all these operators must have to be represented in the selection plan. For example, the proportionate selection operator can be implemented by choosing the i-th solution for its inclusion to P with a probability $f_i / \sum_i f_i$. This probability distribution around B is used to choose choose μ members of P. The ranking selection operator can be implemented with the *sorted ranks* rnk_i and using a probability $rnk_i / \sum_i rnk_i$. The tournament selection operator with a size s can be implemented by using a SP in which s solutions are chosen and the best is placed in P. The above procedure needs to be repeated μ times to create the parent population P.

5.1.3 Niche-preservation and mating restriction operators

A niche-preservation operator is often used to maintain a diverse set of solutions in the population.

Among all the four steps of the proposed algorithm-generator, only the selection plan gets affected to implement a niche-preservation operator. While choosing the parent population P, care should be given to lower the selection probability to population-best solutions of P. Solutions with a wider diversity in their decision variables must be given priorities. The standard sharing function approach (Goldberg and Richardson 1987), clearing approaches (Petrowski 1996), and others can be designed using an appropriate selection plan.

A mating restriction operator, on the other hand, is used to reduce the chance of creating *lethal* solutions arising from mating of two dissimilar yet good solutions. This requires a selection plan SP in which the procedures of choosing of each of the μ parents become dependent to each other. Once the first parent is chosen, the procedure of choosing other parents must consider a similarity measure with respect to the first parent.

We are now ready to discuss the commonly-used models for real-parameter evolutionary optimization.

5.1.4 Generational versus steady-state EAs

Evolutionary algorithms, particularly genetic algorithms, are often used with a generational or with a steady-state concept. In the case of former, a complete population of λ solutions are first created before making any further decision. The proposed algorithm-generator can be used to develop a generational EA by repeatedly using the SP-GP plans to create λ new offspring solutions. Thereafter, the replacement plan simply chooses the whole parent population to be replaced, or $R = B$ (and $r = \mu$). With an elite-preservation operator, the update plan chooses the best μ solutions from the combined population $B \cup C$. In an EA without elitism, the update plan only chooses the complete offspring population C. We describe a generic real-parameter generational GA in detail later.

On the other extreme, a steady-state EA can be designed by using a complete SP-GP-RP-UP cycle for creating and replacing only one solution ($\lambda = r = 1$) in each iteration. It is interesting to note that the SP can use a multi-parent ($\mu > 1$) population, but the GP creates only one offspring solution from it. We shall return to two specific steady-state evolutionary algorithms later. The generational gap EAs can be designed with a non-singleton C and R (or having $\lambda > 1$ and $r > 1$).

The generational model is a direct extension of the canonical binary GAs to real-parameter optimization. In each iteration of this model, a complete set of N new offspring solutions are created. For preserving elite solutions, both the parent and offspring populations are compared and the best N solutions are retained. In most such generational models, the tournament selection (SP) is used to choose two parent solutions and a recombination and a mutation operator are applied to the parent solutions to create two offspring solutions. In terms of our algorithm-generator, the four plans are described below. However, the first two steps are performed iteratively till N offspring solutions are formed.

SP: This is usually a tournament selection, in which two random solutions from B are compared and the better one is selected. By two consecutive tournament selection operations, two parent solutions (P) are chosen.

GP: A real-parameter recombination operator and a real-parameter mutation operator are used to create two offspring solutions ($\lambda = 2$) from P. Continue the above two steps till N offspring solutions (the set C) is formed.

RP: Set $R = B$.

UP: Set B with the best N solutions of $B \cup C$.

The BLX, SBX, and the fuzzy recombination are often used with this generational model. The random mutation operator (Michalewicz 1992), uniform mutation operator (Schwefel 1987), and polynomial mutation operator (Deb and Goyal 1996) are used along with a recombination operator.

Another commonly-used real-parameter genetic algorithm is called the CHC (Eshelman 1991) in which both parent and offspring population (of the same size N) are combined and the best N members are chosen. Such an algorithm can be realized by using a update plan which chooses the best N members from a combined $B \cup C$ population. The CHC algorithm also uses a mating restriction scheme, a matter which can also be implemented using the proposed algorithm-generator as described in Sect.5.1.3

Besides the classical and evolutionary algorithms, there exist a number of hybrid search and optimization methods in which each population member of a GA is undergone with a local search operation (mostly using one of the classical principles). In the so-called Lamarckian approach, the resulting solution vector replaces the starting GA population member, whereas in the Baldwin approach the solution is unchanged but simply the modified objective function value is used in the subsequent search operations. Recent studies (Joines and Houck 1994; Whitley et al. 1994) showed that instead of using a complete Baldwin or a complete Lamarckian approach to all population members, the use of Lamarckian approach to about 20–40% of the population members and the use of Baldwin approach to the remaining solutions is a better strategy. In any case, the use a local search strategy to update a solution can be considered as a special-purpose *mutation* operator associated with the generational plan of the proposed algorithm-generator. Whether the mutated solution is accepted in the population (the Lamarckian approach) or simply the objective function value of the mutated solution is used (the Baldwin approach) or they are used partially (the partial Lamarckian approach) is an implementational matter.

5.1.5 Minimal generation gap (MGG) model

This model is completely different from the generational model. It is a steady-state GA, in which in every iteration only two new solutions are updated in the GA population (Higuchi et al. 2000):

1. From the solution bank B, select μ parents randomly.
2. Generate λ offspring from μ parents using a recombination scheme.
3. Choose two parents at random from the population B.
4. Of these two parents, one is replaced with the best of λ offspring and the other is replaced with a solution chosen by a roulette-wheel selection procedure from a combined population of λ offspring and two chosen parents.

This GA was originally proposed by Satoh, Yamamura and Kobayashi (1996) and later used in a number of studies (Higuchi et al. 2000; Kita et al. 1999; Tsutsui et al. 1999). In the context of our algorithm-generator, the above model can be described as follows:

SP: This uses a uniform probability for choosing any solution from B. In total, μ solutions are picked.
GP: A real-parameter recombination operator (such as SPX or UNDX described in the Appendix) is applied to the set P as many as λ times to create the offspring set C.
RP: R is set by choosing two solutions from B at random. Thus, $r = |R| = 2$.
UP: Here, one solution in R is replaced by the best of C and other by using a roulette-wheel selection on the set $C \cup R$.

The original study considered two different recombination plans (SPX and UNDX). Typical parameter settings used with this algorithm were as follows: $\mu = n + 1$ and $\lambda = 200$ with the SPX recombination operator (Higuchi et al. 2000) and $\mu = 3$ and $\lambda = 200$ with the UNDX recombination operator (kita 1998). Although the developers of this model did not perform a detailed parametric study, we perform a parametric study by varying the offspring size λ on the MGG model with UNDX and SPX operators. The following three commonly-used test problems are used:

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

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

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

First, we fix population size $N = 300$ and vary λ from 2 to 300. All other parameters are kept as they were used in the original MGG study (Higuchi et al. 2000), except that in UNDX $\mu = 6$ is used here, as this value is found to produce better results. In all experiments, we have run the MGG model till a pre-defined number of function evaluations F^T have elapsed. We have used the following values of F^T for different functions: $F^T_{\text{elp}} = 0.5(10^6)$, $F^T_{\text{sch}} = 1(10^6)$ and $F^T_{\text{ros}} = 1(10^6)$. In all experiments, 50 runs with different initial populations are taken and the smallest, median, and highest best function values are recorded. Figure 5 shows the best function values obtained by the SPX and the UNDX operators on F_{elp} with different values of λ. The figure shows that $\lambda = 50$ produced the best reliable performance for the SPX operator. Importantly, the MGG model with $\lambda = 200$ (which was suggested and used in the original MGG study) did not perform as well. Similarly, for the UNDX operator, the best performance is observed at $\lambda = 4$, which is much smaller than the suggested value of 200. The optimality in best function value with respect to λ is clearly evident from the figure. Figure 6 shows that in F_{sch} best performances are observed with identical values for λ with SPX and UNDX. Figure 7 shows the population best objective function value for the MGG model with SPX and UNDX operators applied to the F_{ros} function. Here, the best performance is observed at

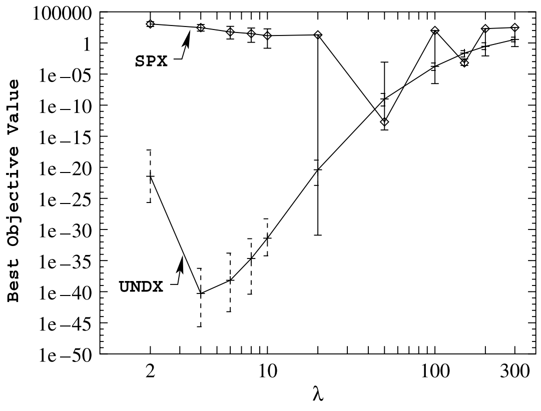

Fig. 5 Best objective function value in B for different λ on F_{elp} using the MGG model with SPX and UNDX operators

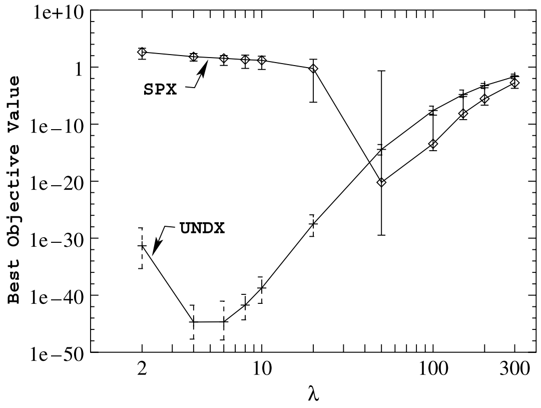

Fig. 6 Best objective function value in B for F_{sch} using the MGG model with SPX and UNDX

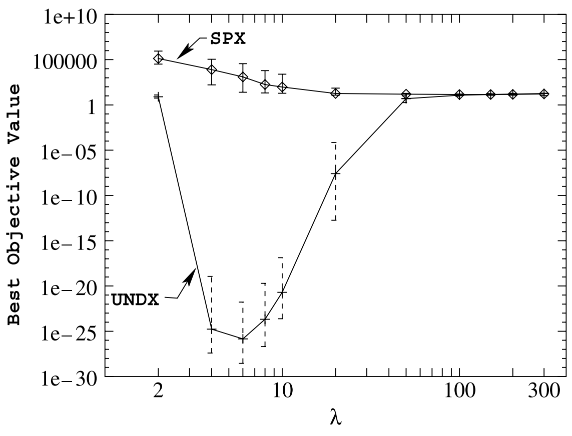

Fig. 7 Best objective function value in B on F_{ros} using the MGG model with SPX and UNDX

$\lambda = 100$ to 300 for the SPX operator and $\lambda = 6$ for the UNDX operator.

Thus, it is clear from the above experiments that the suggested value of $\lambda = 200$ (which was recommended and used in earlier studies (kita et al. 1999; Satoh et al. 1996) is not optimal for either recombination operator (UNDX or SPX). Instead, a smaller value of λ has exhibited better performance. It is also clear from the figures that the SPX operator works better with a large offspring pool size, whereas the UNDX works well with a small offspring pool size. Since a uniform probability distribution on a large simplex is used in the SPX operator, a large pool size requirement is intuitive for a proper sampling of the region covered by the simplex. On the other hand, with a biased probability distribution towards the centroid of the parent solutions, a few samples are enough with the UNDX operator.

However, in order to properly evaluate the MGG model, more such parametric studies must be performed. In this direction, the effect of replacement set size (r) and the parent set size (μ) on the performance of MGG must also be carefully studied. In addition, the MGG model can also be modified with other strategies (such as with a different selection plan (SP) or a different replacement plan (RP) or a different update plan (UP) or other combinations) may also be tried. The representation of the MGG model in the above-mentioned structured manner allows an user to study individual plans separately and understand their interactions.

5.1.6 Generalized generation gap (G3) model

In another study (Deb et al. 2002), the MGG model is modified to make it computationally faster by replacing the roulette-wheel selection with a block selection of the best two solutions. This model also preserves elite solutions from the previous iteration:

1. From the population B of size N, select the best parent and $\mu - 1$ other parents randomly.
2. Generate λ offspring from the chosen μ parents using a recombination scheme.
3. Choose r parents at random from the population B.
4. From a combined subpopulation of r chosen parents and λ created offspring solutions, choose the best r solutions and replace the chosen r solutions (in Step 3) with these solutions.

Although it is clear, for the sake of completeness we rewrite the above model in terms of our algorithm-generator:

SP: The best parent and the $(\mu - 1)$ other random solutions are picked from B.
GP: Create λ offspring solutions from P using any recombination scheme (PCX, UNDX, SPX, or any other recombination operator described in the Appendix).
RP: Set R with r random solutions from B.
UP: From the combined set $C \cup R$, choose r best solutions and put them in R'. Update B as $B := (B \backslash R) \cup R'$.

To compare the performance of the MGG and the G3 model, we consider the UNDX recombination operator as the GP and apply both MGG and G3 methods to three test problems described earlier. In all cases, we use $n = 20$ variables and continue till a function value of 10^{-20} is achieved. Recall that in all three objective functions the minimum function value is zero. But before we describe the simulation results, let us discuss an important matter which we would like to highlight for the future studies on population-based optimization algorithms. All population-based algorithms require the user to supply an initial guess population. In most past studies, this population was initialized symmetrically around the known optimum of the chosen test problem. Along with such an initial population, if a recombination operator which biases solutions in the center of the population is employed, it is not surprising that such an algorithm will work very well. To obtain a real evaluation of an algorithm on a test problem, one way to remedy this difficulty is to use an initial population far away from the known optimum solution. Here, for each test problem, the population $B^{(0)}$ is initialized randomly in $x_i \in [-10, -5]$, which is away from the global optimum solution ($x_i = 0$ for F_{elp} and F_{sch} and $x_i = 1$ for F_{ros}).

Table 1 shows the minimum number of evaluations needed in 50 different replications of each GA. In each case, results are obtained with their best parameter

Table 1 Comparison of MGG and G3 optimization models with the UNDX recombination operator

Function	MGG	G3
F_{elp}	2,97,546	17,826
F_{sch}	5,03,838	30,568
F_{ros}	9,38,544	71,756

settings. In all cases, a population size $N \sim 100$, $\mu = 3$, and λ in the range 2 to 6 produced the best performance. These results indicate that the G3 model is an order of magnitude faster than the MGG model. By keeping the best population member in P and by using a more direct update strategy (instead of using a roulette-wheel selection plan used in MGG), the G3 model makes the search much faster than the MGG model.

5.2 Self-adaptive evolution strategies

Besides the real-parameter genetic algorithms, self-adaptive evolution strategies (ESs) are often used for solving real-parameter optimization problems. In their standard models, a $(\mu/\rho + \lambda)$ or $(\mu/\rho, \lambda)$ self-adaptive ES starts with μ randomly chosen solutions B. Thereafter, by using a recombination scheme (intermediate or discrete) on ρ of the μ parents and a self-adaptive mutation scheme (Beyer 2001), λ (usually greater than μ) offspring solutions (the set C) are created. In the plus strategy, both parent and offspring populations are combined together and the best μ solutions are retained. In the comma strategy, the best μ of λ offspring solutions are retained. In terms of the proposed algorithm-generator, we express these two self-adaptive ESs as follows:

SP: Set $P = B$.
GP: Here, an ES recombination and mutation plan are chosen (see (Bäck 1996; Beyer 2001; Schwefel 1981) for standard operators).
RP: Set $R = B$.
UP: For the plus ES, B is set with the best λ solutions of $R \cup C$ and for the comma ES, it is set with the best λ solutions of C alone.

Elsewhere (Fogel et al. 1995), the evolutionary programming (EP) technique is used in a similar fashion for real-parameter optimization. Those EP techniques can also be represented by our algorithm-generator model in a very similar manner as described above.

5.3 Differential evolution

Differential evolution (DE) (Storn and Price 1997) is a steady-state EA in which for every offspring a set of three parent solutions and an index parent are chosen. Thereafter, a new solution is created either by a variable-wise linear combination of three parent solutions or by simply choosing the variable from the index parent with a probability. The resulting new solution is compared with the index parent and the better of them is declared as the offspring. In the context of our algorithm-generator, the DE can be written as follows:

SP: Here, three solutions and an index solution are chosen at random from a solution bank B. Thus, $\mu = |P| = 4$.
GP: Create an offspring solution (the singleton C) from P with a specified recombination plan (see (Storn and Price 1997) for details and different variations).
RP: Set R with the index parent chosen in SP. Thus, $r = 1$.
UP: Update B by replacing R with the better of R and C.

6 Suggested properties of a generational (recombination) plan

Although the performance of an optimization algorithm depends on the interaction of all four plans outlined in the algorithm-generator, the generational plan of creating new solutions from a set of old solutions is probably the most crucial one to design. In Sect. A, we reviewed a number of generational plans commonly used in the literature. Here, we discuss some salient properties which may be kept in mind while designing a new generational plan (GP).

The task of a GP is to use the members of the set P and create a new pool of solutions (the set C) by exploiting location and spread of solutions in P. To make the plans independent from each other, it is advisable not to use any objective function information explicitly in GP. This is because the emphasis of good solutions based on objective function information was already the task of the selection plan (SP). An objective function based emphasis in both SP and GP may make the resulting algorithm overly greedy, causing a premature convergence in certain problems. Based on this understanding, Beyer and Deb (2001) argued that a recombination operator must have the following two properties:

1. Population mean decision variable vector should remain the same before and after the generational plan is applied.
2. The spread in members of C must be more than that in P.

Since the generational plan does not usually use any objective function information explicitly, the first argument makes sense for global optimization. However, we emphasize the fact that this may not be a strict requirement, particularly in designing an algorithm to find a local optimal solution or to solve a unimodal optimization problem.

The second argument comes from the realization that the selection plan (SP) has a general tendency to reduce the population variance by preferring a few solutions from the solution bank. Thus, the population variance must be increased by the generational plan (GP) to preserve adequate diversity in the offspring population. Figure 8 illustrates this matter. If the GP causes a reduction in the variance, then both SP and GP have continual tendencies of reducing the variance of the population, a matter which will lead the search process towards a premature convergence.

Following the above argument for global optimization, we realize that the population mean of P and C can be preserved by several means. One method would be to have individual recombination events preserving the

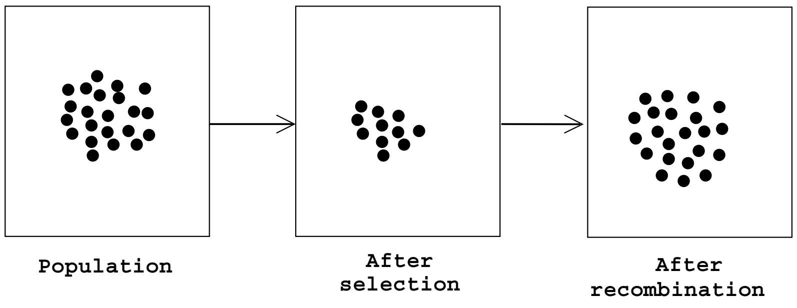

Fig. 8 When the selection plan reduces the diversity of *P*, the tecombination operator must increase the diversity

mean between the participating parents and resulting offspring. We call this approach as the *mean-centric* recombination. The other approach would be to have an individual recombination event biasing offspring to be created near the parents, but assigning each parent an equal probability of creating offspring in its neighborhood. This will also ensure that the population mean of the entire offspring population is identical to that of the parent population. We call this latter approach the *parent-centric* recombination. In the following subsection, we discuss the merits of each of these recombination operators.

6.1 Advantages of parent-centric operators

A couple of examples of the mean-centric recombinations are UNDX and SPX operators, whereas some examples of the parent-centric recombination operators are the PCX, the fuzzy recombination, and the SBX operator. It is argued here that parent-centric recombinations are, in general, better than the mean-centric ones. Since each parent is carefully picked by the selection plan (meaning that the parents are preferred solutions from the solution bank *B*), for most real-parameter optimization problems it can be assumed that solutions close to these parents are also likely to be potential good candidates. On the contrary, it may be quite demanding to assume that the solutions close to the centroid of the participating parents are also good, especially in cases where parents are well sparsed in the search space. This situation happens in the early iterations of an algorithm, when most population members are randomly placed over the entire search space. Although individual parents may be good, but the resulting centroid of the parents need not be good as well. Thus, emphasizing solutions close to the centroid of the parents (as enforced in UNDX directly and in SPX indirectly) may not be a good idea. However, creating solutions close to parents as emphasized by the PCX operator should make a more steady and reliable search. An earlier study (Beyer and Deb 2001) has demonstrated that a variable-wise parent-centric recombination operator (SBX) produced a faster convergence towards the true optimum on a number of test problems compared to a uniformly emphasized recombination operator (BLX).

7 A parametric study of the G3 model

In this section, we suggest a modified version of the PCX operator originally suggested by the author and his students (Deb et al. 2002) and then compare its performance with respect to a representative mean-centric recombination operator (UNDX) on the G3 model. Finally, the performance of the G3 model with the modified PCX and the original PCX operators is compared with other population-based optimization algorithms, such as a classical quasi-Newton method, three self-adaptive ESs, and the differential evolution strategy.

7.1 A modified PCX operator

In the original PCX operator, there exists a finite (albeit small) probability of creating an offspring solution on the far side of the centroid. Figure 9(a) shows a typical probability distribution of PCX on solution 1 for creating offspring solutions. When all parent solutions are to be used one after another with

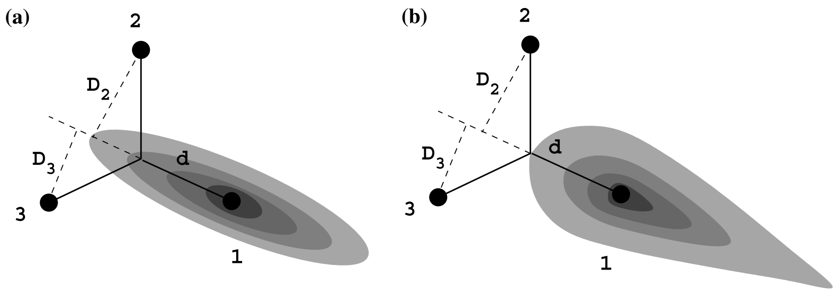

Fig. 9 The original and the modified PCX operators are illustrated in **a** and **b**, respectively

the original PCX operator, this causes an artificial increase in probability of creating solutions near the centroid. In the modified PCX operator, we restrict the offspring to be created strictly on the region in which the parent lies. Figure 9(b) illustrates this modification (mPCX). For this purpose, we simply suggest the use of a log-normal probability distribution for the $\vec{d}^{(p)}$ component, as given below:

$$\vec{y} = \vec{x}_p + (\exp(w_\zeta) - 1)\vec{d}^{(p)} + \sum_{i=1,\, i\neq p}^{\mu} w_\eta \bar{D}\vec{e}^{(i)}, \qquad (4)$$

The parameter w_ζ is a zero-mean normally distributed variable with standard deviation $\sqrt{2\log\sigma_\zeta}$. This modified PCX operator ensures a zero probability of creating a solution on the other side of the centroid.

7.2 Comparison with the original PCX operator

Figure 10 shows the evaluations needed to achieve a function value of 10^{-20} using the G3 model with the modified PCX, original PCX, and the UNDX recombination operators. In all cases, a population size $N = 100$ and a parent size $\mu = 3$ are used. For the mPCX operator, we have used $\sigma_\eta = 0.1$ and $\sigma_\zeta = 1.01$. However, for the PCX operator, we use $\sigma_\eta = \sigma_\zeta = 0.1$. For the UNDX operator, we use the suggested values (Kita and Yamamura 1999). The figure clearly shows that the G3 model with the PCX operators performs much better than that with the UNDX operator. For most λ values, the modified PCX operator produces slightly better results than the original PCX operator. The smallest number of evaluations needed by the mPCX operator is 5,194 (with $\lambda = 2$ and $N = 100$), whereas the smallest number of evaluations needed by the original PCX was 5,818 with identical λ and N values. It is also interesting to note from this parametric study that the performance of G3 gets better with reducing λ, irrespective of the recombination operator used. Since a fixed number of evaluations is allowed in each run, the G3 model works better with more iterations and fewer new solutions created in each iteration.

In order to investigate the effect of population size N on the performance of the three recombination operators, we fix the offspring set size to $\lambda = 2$ and vary N. Although a more detailed statistical test is necessary, a visual comparison shown in Fig. 11 reveals that the mPCX operator performs better than original PCX for smaller N and the best performance occurs with a population size of 100. This study indicates the importance of such a parametric study in developing an efficient algorithm. The study reveals that there exists an optimal range of population size for which the G3 model works the best. For all three recombination operators, a population size around 100 performs well.

Figure 12 shows the performance comparison for the 20-variable Schwefel's function. It is also clear from

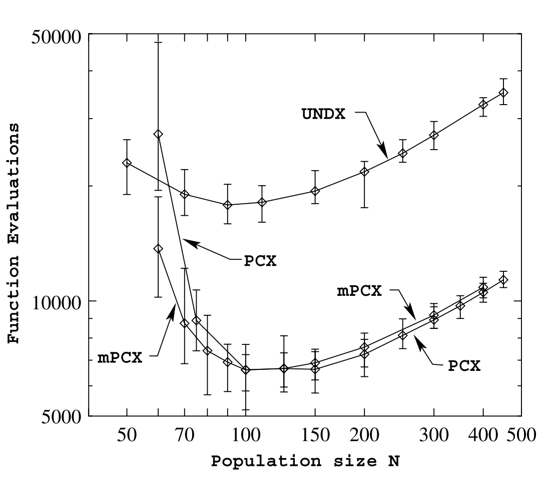

Fig. 11 Function evaluations versus population sizes on F_{elp} using the G3 model with mPCX, PCX and UNDX. $\lambda = 2$ is used

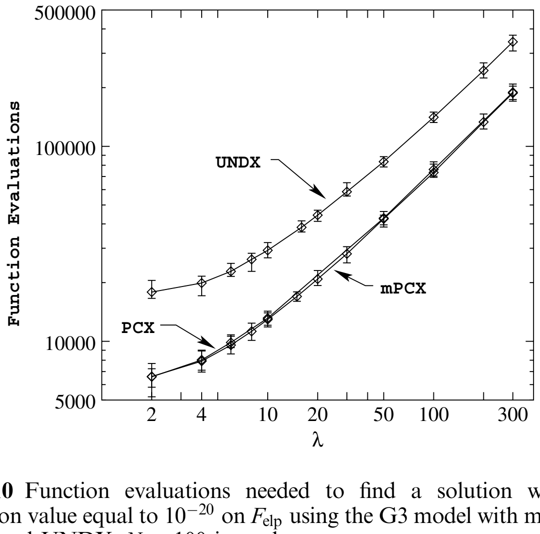

Fig. 10 Function evaluations needed to find a solution with a function value equal to 10^{-20} on F_{elp} using the G3 model with mPCX, PCX and UNDX. $N = 100$ is used

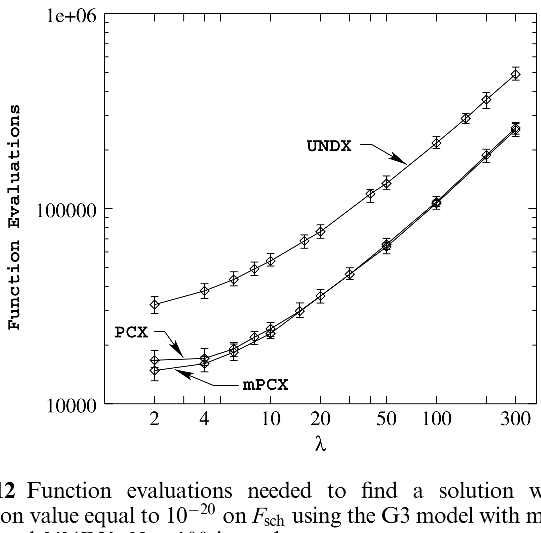

Fig. 12 Function evaluations needed to find a solution with a function value equal to 10^{-20} on F_{sch} using the G3 model with mPCX, PCX and UNDX. $N = 100$ is used

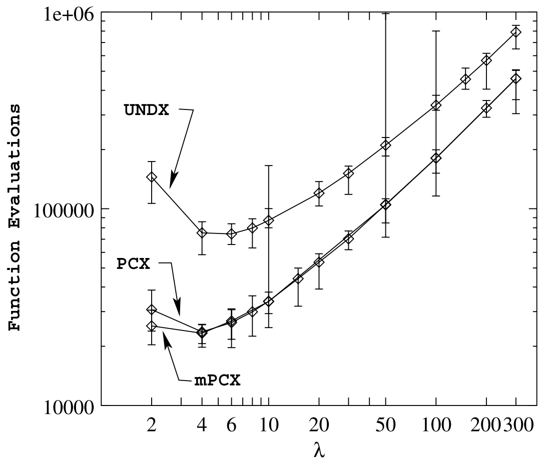

Fig. 13 Function evaluations needed to find a solution with a function value equal to 10^{-20} on F_{ros} using the G3 model with mPCX, PCX and UNDX. $N = 100$ is used

the figure that the modified PCX operator performs slightly better than the original PCX operator even in this function. For a population size of $N = 100$, the best performance for mPCX is recorded with $\lambda = 2$ requiring 13,126 evaluations compared to 15,358 required with the original PCX. For this function also, the performance of G3 model with any of the three recombination operators is better with smaller offspring size (λ). We have also observed an optimal population size (N) for this function. For brevity, we do not show the plot here.

Figure 13 shows the performance comparison for the 20-variable Rosenbrock's function. A slightly better performance of the modified PCX operator is observed again. In this function, the best performance (requiring 19,692 evaluations) with $N = 100$ is observed for $\lambda = 6$, compared to 20,640 evaluations needed with the original PCX operator having $\lambda = 4$. It is interesting to note that for this function the G3 model with all three recombination operators performs the best with λ around 4 to 6. For λ smaller than these values, the performance deteriorates. As the problem gets more complicated, an adequate number of offspring samples are needed to provide useful information about the search space. Once again, the performance of the G3 model is found to be optimal for a specific range of N in this function. We do not present the detail results here for brevity.

7.3 Comparison with other evolutionary algorithms

Next, we compare the G3 model and both PCX operators with other evolutionary optimization methods. Table 2 shows the total function evaluations needed by various algorithms to find a solution with a function value equal to or smaller than 10^{-20} for all three test problems. In all cases, 50 different runs are made and the best, median and the worst total evaluations of the 50 runs are recorded. It is clear that for both elliptical and Schwefel's functions, the G3 + mPCX works the best. However, in all three test problems, G3 model with either PCX operators performs much better than other evolutionary methods. In all cases except the CMA-ES (Hansen and Ostermeier 1996)– an evolution strategy which works by deriving salient search directions from a collection of previously-found good solutions, the performance is at least an order of magnitude better. Although the CMA-ES approach comes closer to the G3 and PCX operators, the latter approaches are better in all three test problems.

7.4 Comparison with a classical optimization method

Finally, we compare the G3 + PCX operator with a classical point-based optimization algorithm – the quasi-Newton method. In Table 3, we present the best, median, and the worst function values obtained from a set of 10 independent runs started from random solutions in $x_i \in [-10, -5]$. The maximum number of function evaluations allowed in each test problem is determined from the best and worst function evaluations needed for the G3 model with PCX operator to achieve an accuracy of 10^{-20}. These limits on function evaluations are also tabulated. The tolerances in the variables and in the function values are set to 10^{-30}. The table shows that the quasi-Newton method has outperformed the G3 model with PCX for the ellipsoidal function by achieving a better function value. Since the ellipsoidal function has no linkage among variables, the performance of the quasi-Newton search method (a point-by-point approach) is difficult to match. However, it is clear from the table that the quasi-Newton method is not able to find the optimum with an accuracy of 10^{-20} (obtained by G3 + PCX) within the allowed number of function evaluations in more epistatic problems (Schwefel's and Rosenbrock's functions).

Table 2 Comparison of G3 model and the modified PCX with a number of other optimization algorithms on F_{elp}, F_{sch} and F_{ros}

EA	F_{elp}			F_{sch}			F_{ros}		
	Best	Median	Worst	Best	Median	Worst	Best	Median	Worst
G3 + mPCX	5,194	6,576	7,240	13,126	14,820	16,708	19,776	23,296	25,852
G3 + PCX	5,826	6,800	7,728	13,988	15,602	17,188	20,640	23,688	25,720
CMA-ES	8,064	8,472	8,868	15,096	15,672	16,464	29,208	33,048	41,076
(1, 10)-ES	28,030	40,850	87,070	72,330	105,630	212,870	591,400	803,800	997,500
(15, 100)-ES	83,200	108,400	135,900	173,100	217,200	269,500	663,760	837,840	936,930
DE	9,660	12,033	20,881	102,000	119,170	185,590	243,800	587,920	942,040

Table 3 Solution accuracy obtained using the quasi-Newton method. FE denotes the maximum allowed function evaluations.

Func.	FE	Best	Median	Worst
F_{elp}	6,000	$8.819(10^{-24})$	$9.718(10^{-24})$	$2.226(10^{-23})$
F_{sch}	15,000	$4.118(10^{-12})$	$1.021(10^{-10})$	$7.422(10^{-9})$
F_{ros}	15,000	$6.077(10^{-17})$	$4.046(10^{-10})$	3.987
F_{elp}	8,000	$5.994(10^{-24})$	$1.038(10^{-23})$	$2.226(10^{-23})$
F_{sch}	18,000	$4.118(10^{-12})$	$4.132(10^{-11})$	$7.422(10^{-9})$
F_{ros}	26,000	$6.077(10^{-17})$	$4.046(10^{-10})$	3.987

8 Conclusions

Real-parameter optimization using classical gradient or direct search algorithms is not new. Although there exist a plethora of point-based algorithms, requiring only one solution in each iteration, there also exist a few population-based algorithms. On the contrary, the real-parameter optimization using population-based evolutionary algorithms are comparatively new. In this paper, we have suggested a population-based real-parameter optimization algorithm-generator, which can represent most classical and evolutionary algorithms. For a number of such existing optimization algorithms, we have demonstrated how the proposed algorithm-generator can be used to express them.

Besides being able to represent an existing algorithm, the structure of the algorithm-generator also provides an ideal platform to investigate different new plans. In such an effort, we have also suggested an efficient algorithm (we called generalized generation gap or G3) and performed a detailed parametric study. To evaluate the proposed G3 algorithm, we have compared its performance with an efficient classical method and a number of other evolutionary optimization algorithms.

The following conclusions can be derived out of this study:

- For real-parameter optimization using a population-based algorithm, an algorithm-generator is proposed. Most population-based optimization algorithms found in classical literature and in evolutionary optimization literature can be easily described using the proposed algorithm-generator.
- The algorithm-generator contains four functional plans, each of which can be independently controlled by the user. This *functional decomposition* of an algorithm allows an user to investigate effect of each functional plan on the algorithm's performance and should help an user to develop efficient optimization algorithms.
- Based on extensive simulation results on three test problems, it has been observed that the G3 model with a parent-centric recombination operator is better than all other population-based real-parameter optimization algorithms considered in this study.
- A parametric study with the population size of B has shown that in all problems the G3 model performs the best only when an adequate population size is considered. This amply emphasizes the need of using population-based approaches (instead of a point-based approach) in solving search and optimization problems efficiently.
- Parent-centric recombination is found to be much better than the mean-centric recombination operators with the G3 model.
- Contrary to many real-parameter optimization studies, it has been emphasized here and in other studies of the author (Deb et al. 2002) that for a proper evaluation of an algorithm the initial population must *not* be created centered around the known optimum of a test problem. This may introduce a inherent bias for evolutionary algorithms with certain generational plans and may not result in a fair comparison with classical and other evolutionary algorithms.
- One other important aspect of this study is that both classical and evolutionary algorithms are treated as an outcome of an identical algorithm-generator. Although evolutionary algorithmists tend to use terminologies which are based on natural genetics and natural evolution, in terms of the proposed algorithm-generator they can be treated differently and probably more meaningfully along the lines of other optimization algorithms.

The algorithm-generator suggested here provides a platform for developing new and hopefully better optimization algorithms. After developing such algorithms, they can be evaluated by performing a parametric study similar to the one performed here with the G3 model. Although this study has considered only three test problems, more test problems and test problems with further complexities must also be tried. Because of the demonstrated common principles between classical and evolutionary optimization algorithms through the proposed algorithm-generator, this study should encourage researchers from both fields to understand the merits and demerits of their approaches and help develop more efficient hybrid algorithms.

A real-parameter recombination operators

The early studies on real-parameter recombination operators concentrated on developing variable-wise operators, in which a recombination is performed on each decision variable from two or more parent vectors at a time with a probability.

One of the earliest implementations was reported by Wright (1991), where a *linear crossover* operator created the three specific solutions from two parent solutions $x_i^{(1,t)}$ and $x_i^{(2,t)}$ at iteration t:

$$0.5(x_i^{(1,t)}+x_i^{(2,t)}),\ (1.5x_i^{(1,t)}-0.5x_i^{(2,t)}),(-0.5x_i^{(1,t)}+1.5x_i^{(2,t)}).$$

Of these three solutions, the best two solutions were chosen to replace the parents. It is clear that the creation of only three pre-destined offspring from two parent solutions does not quite make this recombination operator efficient to be used.

Following an *interval schema* processing suggested by Eshelman and Schaffer (1993) (which is incidentally similar in concept to Goldberg's (1991) *virtual alphabets*), they suggested a blend crossover or BLX-α operator for real-parameter GAs. For two parent solutions $x_i^{(1,t)}$ and $x_i^{(2,t)}$ (assuming $x_i^{(1,t)} < x_i^{(2,t)}$), the BLX-$\alpha$ randomly picks a solution in the range $[x_i^{(1,t)} - \alpha(x_i^{(2,t)} - x_i^{(1,t)}), x_i^{(2,t)} + \alpha(x_i^{(2,t)} - x_i^{(1,t)})]$. This crossover operator is illustrated in Fig.14. Thus, if u_i is a random number between 0 and 1, the following is an offspring:

$$x_i^{(1,t+1)} = (1-\gamma_i)x_i^{(1,t)} + \gamma_i x_i^{(2,t)}, \tag{5}$$

where $\gamma_i = (1+2\alpha)u_i - \alpha$. However, it is important to note that the factor γ_i is uniformly distributed for a fixed value of α. The figure also depicts this fact. The range of this probability distribution depends on the parameter α. If α is chosen to be zero, this crossover creates a random solution in the range $(x_i^{(1,t)}, x_i^{(2,t)})$. In a number of test problems, the investigators have reported that BLX-0.5 (with $\alpha = 0.5$) performs better than BLX operators with any other α value. It is important to note that there exists a number of other crossover operators which work by using the same principle. The arithmetic crossover (Michalewicz and Janikow, 1991) uses Eq.(5) with a fixed value of γ for all decision variables. However, γ is chosen by carefully calculating its maximum allowed value in all decision variables so that the resulting offspring does not exceed the lower or upper limits. The extended crossover operator (Voigt et al. 1995) is also similar to BLX-α.

The simulated binary recombination (SBX) operator assigns more probability for an offspring to remain closer to the parents than away from parents (Deb and Agrawal 1995). The procedure of computing the offspring $x_i^{(1,t+1)}$ and $x_i^{(2,t+1)}$ from the parent solutions $x_i^{(1,t)}$ and $x_i^{(2,t)}$ is described as follows. A spread factor β_i is defined as the ratio of the absolute difference in offspring values to that of the parents:

$$\beta_i = \left| \frac{x_i^{(2,t+1)} - x_i^{(1,t+1)}}{x_i^{(2,t)} - x_i^{(1,t)}} \right|. \tag{6}$$

First, a random number u_i between 0 and 1 is created for the i-th variable. Thereafter, from a specified probability distribution function, the ordinate β_{q_i} is found so that the area under the probability curve from 0 to β_{q_i} is equal to the chosen random number u_i. The probability distribution is chosen to have a search power similar to that in a single-point crossover in binary-coded GAs and is given as follows (Deb and Agrawal 1995):

$$\mathscr{P}(\beta_i) = \begin{cases} 0.5(\eta_c+1)\beta_i^{\eta_c}, & \text{if } \beta_i \le 1; \\ 0.5(\eta_c+1)\frac{1}{\beta_i^{\eta_c+2}}, & \text{otherwise.} \end{cases} \tag{7}$$

Figure 15 shows this probability distribution with $\eta_c = 2$ and 5 for creating offspring from two parent solutions ($x_i^{(1,t)} = 2.0$ and $x_i^{(2,t)} = 5.0$) in the real space. In the above expressions, the distribution index η_c is any non-negative real number. A large value of η_c gives a higher probability for creating 'near-parent' solutions and a small value of η_c allows distant solutions to be selected as offspring. Typically, η_c in the range of 2 to 5 is used in most single-objective optimization studies (Deb and Goyal 1998). After obtaining β_{qi} from the above probability distribution, two offspring solutions are calculated as follows:

$$x_i^{(1,t+1)} = 0.5\Big[(1+\beta_{q_i})x_i^{(1,t)} + (1-\beta_{q_i})x_i^{(2,t)}\Big], \tag{8}$$

$$x_i^{(2,t+1)} = 0.5\Big[(1-\beta_{q_i})x_i^{(1,t)} + (1+\beta_{q_i})x_i^{(2,t)}\Big]. \tag{9}$$

Note that two offspring are symmetrically placed around the parent solutions. This is deliberately enforced to avoid a bias towards any particular parent solution in a single recombination operation. Another interesting aspect of this recombination operator is that for a fixed η_c the offspring have a spread which is proportional to that of the parent solutions. From Eqs. 8 and 9, we have

$$\Big(x_i^{(2,t+1)} - x_i^{(1,t+1)}\Big) = \beta_{q_i}\Big(x_i^{(2,t)} - x_i^{(1,t)}\Big). \tag{10}$$

This has an important implication. If the difference in decision variable values of parents (the term inside the bracket in the right side term) is smaller (larger), the offspring is also closer (more distant) to the parents. Thus, during early iterations, parents being far away from each other, offspring are also created on the entire search space, thereby providing a good initial search of the entire space. When solutions converge near a good region, the parents are closer to each other and this

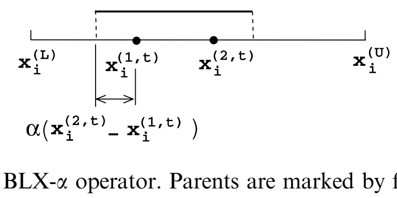

Fig. 14 The BLX-α operator. Parents are marked by filled circles

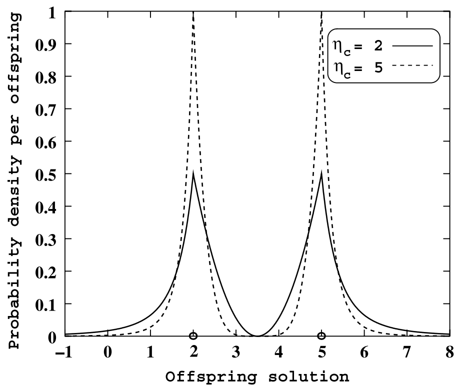

Fig. 15 The probability density function for creating offspring under an SBX-η_c operator. Parents are marked with an 'o'

operator helps provide a more focused search. This property makes the resulting GA self-adaptive (Deb and Beyer 2001).

Voigt et al. (1995) suggested a fuzzy recombination (FR) operator, which is similar to the SBX operator. In some sense, the FR operator can be thought of as a special case of the SBX operator with $\eta_c = 1$.

The fuzzy connectives based recombination operator (Herrara et al. 1997) is also applied variable by variable. For each variable x_i, three regions are marked: region I $(x_i^{(L)}, x_i^{(1,t)})$, region II $(x_i^{(1,t)}, x_i^{(2,t)})$ and region III $(x_i^{(2,t)}, x_i^{(U)})$ (here, we assume that $x_i^{(1,t)} < x_i^{(2,t)}$). The fourth region is constructed as an overlapping region $(y_i^{(1)}, y_i^{(2)})$ with all of the above three regions. Here, $y_i^{(1)} \le x_i^{(1,t)}$ and $y_i^{(2)} \ge x_i^{(2,t)}$. Once these regions are identified, one solution is chosen from each of them by using two user-defined fuzzy connective functions, which are defined over the two normalized parents.

Besides, there exists a number of other recombination operators such as the unfair average crossover (Nomura and Miyoshi 1996) and extensions to the above operators. Although these operators are well-tested on different problems, because of their strict variable-wise application they may not be adequate in handling problems which requires *linkage preservation* among variables (Harik and Goldberg 1996). To be able to solve such problems, researchers have also suggested a number of recombination operators which directly work on the decision variable vectors. In the following paragraphs, we describe a few such operators.

In the unimodal normally distributed recombination (UNDX) operator (Ono and Kobayashi 1997), $(\mu - 1)$ parents are randomly chosen and their mean $\vec{g}$ is computed. From this mean, the $(\mu - 1)$ direction vectors $\vec{d}^{(i)} = \vec{x}^{(i)} - \vec{g}$ is formed. Let the direction cosines be $\vec{e}^{(i)} = \vec{d}^{(i)}/|\vec{d}^{(i)}|$. Thereafter, from another randomly chosen parent $\vec{x}^{(\mu)}$, the length D of the vector $(\vec{x}^{(\mu)} - \vec{g})$ orthogonal to all $\vec{e}^{(i)}$ is computed. Let $\vec{e}^{(j)}$ (for $j = \mu, \ldots, n$, where n is the size of the variable vector $\vec{x}$) be the orthonormal basis of the subspace orthogonal to the subspace spanned by all $\vec{e}^{(i)}$ for $i = 1, \ldots, (\mu - 1)$. Then, the offspring is created as follows:

$$\vec{y} = \vec{g} + \sum_{i=1}^{\mu-1} w_i |\vec{d}^{(i)}| \vec{e}^{(i)} + \sum_{i=\mu}^{n} v_i D \vec{e}^{(i)}, \tag{11}$$

where w_i and v_i are zero-mean normally distributed variables with variances σ_ζ^2 and σ_η^2, respectively. Kita and Yamamura (1999) suggested $\sigma_\zeta = 1/\sqrt{\mu - 2}$ and $\sigma_\eta = 0.35/\sqrt{n - \mu - 2}$, respectively and observed that $\mu = 3$ to 7 performed well. It is interesting to note that each offspring is created around the mean vector $\vec{g}$. The probability of creating an offspring away from the mean vector reduces and the maximum probability is assigned at the mean vector. Figure 16 shows three parents and a few offspring created by the UNDX operator. The complexity of the above procedure in creating one offspring is $O(\mu^2)$, governed by the Gram-Schmidt orthonormalization needed in the process.

The simplex recombination (SPX) operator (Tsutsui et al. 1999) also creates offspring around the mean, but restricts them within a predefined region (in a simplex similar but $\gamma = \sqrt{\mu + 1}$ times bigger than the parent simplex). A distinguishing aspect of SPX from UNDX operator is that the SPX assigns a uniform probability distribution for creating any solution in a restricted region. Figure 17 shows a number of offspring solutions created using three parents. The computational complexity for creating one offspring here is $O(\mu)$.

The parent-centric recombination (PCX) operator (Deb et al. 2002) is an extension of the SBX operator (described earlier) for any number of parents. For μ parents, first the mean vector $\vec{g}$ is computed. Thereafter, for each offspring, one parent $\vec{x}^{(p)}$ is chosen with equal probability. The direction vector $\vec{d}^{(p)} = \vec{x}^{(p)} - \vec{g}$ is calculated. Thereafter, from each of the other $(\mu - 1)$ parents perpendicular distances D_i to the line $\vec{d}^{(p)}$ are computed and their average $\bar{D}$ is found. The offspring is created as follows:

$$\vec{y} = \vec{x}_p + w_\zeta \vec{d}^{(p)} + \sum_{i=1,\, i\neq p}^{\mu} w_\eta \bar{D} \vec{e}^{(i)}, \tag{12}$$

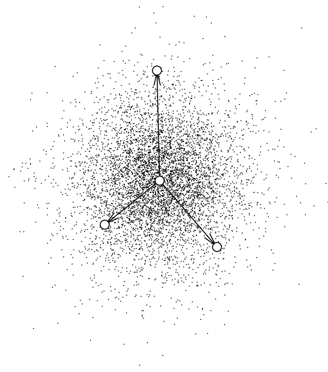
Fig.16 UNDX

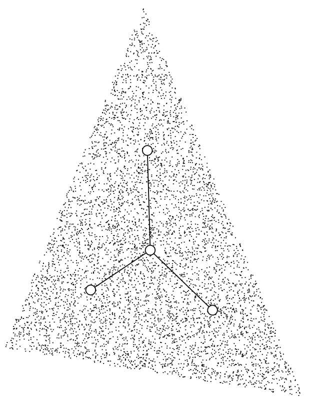
Fig.17 SPX

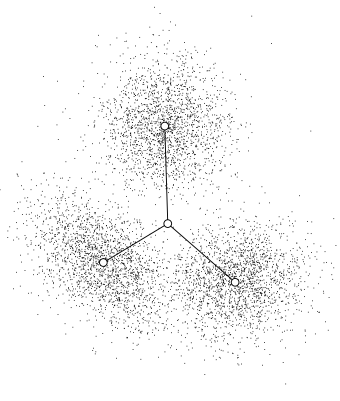

Fig. 18 PCX

where $\vec{e}^{(i)}$ are the $(\mu - 1)$ orthonormal bases that span the subspace perpendicular to $\vec{d}^{(p)}$. Thus, the complexity of the PCX operator to create one offspring is $O(\mu)$, instead of $O(\mu^2)$ required for the UNDX operator. The parameters w_ζ and w_η are zero-mean normally distributed variables with variance σ_ζ^2 and σ_η^2, respectively. The important distinction from the UNDX operator is that offspring solutions are centered around each parent under the PCX operator. The probability of creating an offspring closer to the parent is more. Figure 18 shows a distribution of offspring solutions with three parents.

Acknowledgement The author wishes to thank his students Dhiraj Joshi, Ashish Anand, Abhishek Porwal, and Amit Gautam for their programming expertise.

References

Bäck T (1996) Evolutionary algorithms in theory and practice. Oxford University Press, New York

Beyer H-G (2001) The theory of evolution strategies. Springer, Berlin Germany

Beyer H-G, Deb K (2001) On self-adaptive features in real-parameter evolutionary algorithms. IEEE Trans Evol Comput 5(3): 250–270

Box GEP (1957) Evolutionary operation: method for increasing industrial productivity. Appl Stat 6(2): 81–101

Box MJ (1965) A new method of constrained optimization and a comparison with other methods. Comput J 8(1): 42–52

Brooks SH (1958) A discussion of random methods for seeking maxima. Operations Res 6: 244–253

Deb K (1995) Optimization for engineering design: algorithms and examples. Prentice-Hall, New Delhi

Deb K (2000) An efficient constraint handling method for genetic algorithms. Comput Methods Appl Mech Eng 186(2–4): 311–338

Deb K (2001) Multi-objective optimization using evolutionary algorithms. Chichester Wiley, UK

Deb K (2003) Unveiling innovative design principles by means of multiple conflicting objectives. Eng Optimization 35(5): 445–470

Deb K, Agrawal RB (1995) Simulated binary crossover for continuous search space. Complex Syst 9(2): 115–148

Deb K, Anand A, Joshi D (2002) A computationally efficient evolutionary algorithm for real-parameter optimization. Evol Comput J 10(4): 371–395

Deb K, Beyer H-G (2001) Self-adaptive genetic algorithms with simulated binary crossover. Evol Comput J 9(2): 197–221

Deb K, Goyal M (1996) A combined genetic adaptive search (GeneAS) for engineering design. Comput Sci Inform 26(4): 30–45

Deb K, Goyal M (1998) A robust optimization procedure for mechanical component design based on genetic adaptive search. Trans ASME: J Mech Des 120(2): 162–164

Duan QY, Gupta VK, Sorooshian S (1993) Shuffled complex evolution approach for effective and efficient global minimization. J Optimization: Theory Appl 76(3): 501–521

Eshelman LJ (1991) The CHC adaptive search algorithm: how to have safe search when engaging in nontradtional genetic recombination. In: Foundations of genetic algorithms 1 (FOGA-1), pp 265–283

Eshelman LJ, Schaffer JD (1993) Real-coded genetic algorithms and interval-schemata. In: Foundations of Genetic Algorithms 2 (FOGA-2), pp 187–202

Fogel DB (1995) Evolutionary Computation. Piscataway, IEEE Press, NY

Fogel LJ, Angeline PJ, Fogel DB (1995) An evolutionary programming approach to self-adaptation on finite state machines. In: Proceedings of the fourth international conference on evolutionary programming, pp 355–365

Goldberg DE (1989) Genetic algorithms for search, optimization, and machine learning. Reading, Addison-Wesley, MA

Goldberg DE (1991) Real-coded genetic algorithms, virtual alphabets, and blocking. Complex Syst 5(2): 139–168

Goldberg DE, Deb K, Thierens D (1993) Toward a better understanding of mixing in genetic algorithms. J Society Instrum Control Eng (SICE) 32(1): 10–16

Goldberg DE, Richardson J (1987) Genetic algorithms with sharing for multimodal function optimization. In: Proceedings of the first international conference on genetic algorithms and their applications, pp 41–49

Hansen N, Ostermeier A (1996) Adapting arbitrary normal mutation distributions in evolution strageties: the covariance matrix adaptation. In: Proceedings of the IEEE international conference on evolutionary computation, pp 312–317

Harik G, Goldberg DE (1996) Learning linkages. In:Foundations of genetic algorithms 4 (FOGA-4), pp 247–262

Herrara F, Lozano M, Verdegay JL (1997) Fuzzy connectives based crossover operators to model genetic algorithms population diversity. Fuzzy Set and Syst 92(1): 21–30

Higuchi T, Tsutsui S, Yamamura M (2000) Theoretical analysis of simplex crossover for real-coded genetic algorithms. In: Parallel problem solving from nature (PPSN-VI), pp 365–374

Holland JH (1975) Adaptation in natural and artificial systems. Ann Arbor MIT Press, MI

Joines JA, Houck CR (1994) On the use of nonstationary penalty functions to solve nonlinear constrained optimization problems with GAs. In: Proceedings of the international conference on evolutionary computation, pp 579–584

Kita H (1998) A comparison study on self-adaptation in evolution strategies and real-coded genetic algorithms. Department of computational intelligence and systems science, Tokyo institute of technology, Japan

Kita H, Ono I, Kobayashi S (1999) Multi-parental extension of the unimodal normal distribution crossover for real-coded genetic algorithms. In: Proceedings of the 1999 congress on evolutionary computation, pp 1581–1587

Kita H, Yamamura M (1999) A functional specialization hypothesis for designing genetic algorithms. In:Proceedings of the 1999 IEEE international conference on systems, man, and cybernetics, pp 579–584

Luus R, Jaakola THI (1973) Optmization by direct search and systematic reduction of the size of search region. AIChE J 19: 760–766

Michalewicz Z (1992) Genetic algorithms + data structures = evolution programs. Springer Berlin

Michalewicz Z, Janikow CZ (1991) Handling constraints in genetic algorithms. In: Proceedings of the fourth international conference on genetic algorithms, pp 151–157

Nelder JA, Mead R (1965) A simplex method for function minimization. Comput J 7: 308–313

Nomura T, Miyoshi T (1996) Numerical coding and unfair average crossover in GA for fuzzy rule extraction in dynamic environments. In: Uchikawa Y, Furuhashi T (eds), Fuzzy logic, neural networks, and evolutionary computation, (lecture notes in computer science 1152). Springer Berlin, pp 55–72

Ono I, Kobayashi S (1997) A real-coded genetic algorithm for function optimization using unimodal normal distribution crossover. In: Proceedings of the seventh international conference on genetic algorithms (ICGA-7), pp 246–253

Petrowski A (1996) A clearing procedure as a niching method for genetic algorithms. In: Proceedings of third IEEE international conference on evolutionary computation(ICEC'96). IEEE press, Piscataway NJ, pp 798–803

Price WL (1983) Global optimization by controlled random search. J Optimization: Theory Appl 40: 333–348

Rao SS (1984) Optimization: theory and applications. Wiley, New York

Rechenberg I (1965) Cybernetic solution path of an experimental problem. Royal aircraft establishment, library translation number 1122, Farnborough

Reklaitis GV, Ravindran A, Ragsdell KM (1983) Engineering optimization methods and applications. Wiley, New York

Rudolph G (1994) Convergence analysis of canonical genetic algorithms. IEEE Trans Neural Netw 5(1): 96–101

Satoh H, Yamamura M, Kobayashi S (1996) Minimal generation gap model for gas considering both exploration and exploitation. In: Proceedings of the IIZUKA: methodologies for the conception, design, and application of intelligent systems, pp 494–497

Schwefel H-P (1981) Numerical optimization of computer models. Chichester Wiley, UK

Schwefel H-P (1987) Collective phenomena in evolutionary systems. In:Checkland P, Kiss I (eds), Problems of constancy and change – the complementarity of systems approaches to complexity. pp 1025–1033. Budapest: International Society for General System Research.

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

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

Thierens D, Goldberg DE (1993) Mixing in genetic algorithms. In:Proceedings of the fifth international conference on genetic algorithms, pp 38–45

Tsutsui S, Yamamura M, Higuchi T (1999) Multi-parent recombination with simplex crossover in real-coded genetic algorithms. In: Proceedings of the genetic and evolutionary computation conference (GECCO-99), pp 657–664

Voigt H-M, Mühlenbein H, Cvetković D (1995) Fuzzy recombination for the breeder genetic algorithm. In:Proceedings of the sixth international conference on genetic algorithms, pp 104–111

Whitley D, Gordon S, Mathias K (1994) Lamarckian evolution, the Baldwin effect and function optimization. In:Parallel problem solving from nature (PPSN-III), pp 6–15

Wright A (1991) Genetic algorithms for real parameter optimization. In: Foundations of genetic algorithms 1 (FOGA-1), pp 205–218