Self-Adaptive Simulated Binary Crossover for Real-Parameter Optimization

Kalyanmoy Deb
Dept. of Mechanical Engineering
Indian Inst. of Tech. Kanpur
Kanpur, PIN 208016, India
deb@iitk.ac.in

S. Karthik
Dept. of Mechanical Engineering
Indian Inst. of Tech. Kanpur
Kanpur, PIN 208016, India
ksindhya@iitk.ac.in

Tatsuya Okabe
Honda Research Institute Japan
8-1 Honcho, Wako-shi
Saitama, 351-0188, Japan
okabe@jp.honda-ri.com

ABSTRACT

Simulated binary crossover (SBX) is a real-parameter recombination operator which is commonly used in the evolutionary algorithm (EA) literature. The operator involves a parameter which dictates the spread of offspring solutions vis-a-vis that of the parent solutions. In all applications of SBX so far, researchers have kept a fixed value throughout a simulation run. In this paper, we suggest a self-adaptive procedure of updating the parameter so as to allow a smooth navigation over the function landscape with iteration. Some basic principles of classical optimization literature are utilized for this purpose. The resulting EAs are found to produce remarkable and much better results compared to the original operator having a fixed value of the parameter. Studies on both single and multiple objective optimization problems are made with success.

Categories and Subject Descriptors

I.2.8 [**Computing Methodologies**]: Problem Solving, Control Methods, and Search

General Terms

Algorithms

Keywords

Self-adaptation, simulated binary crossover, real-parameter optimization, recombination operator.

1. INTRODUCTION

Most real-world optimization problems involve decision variables which are real-valued. Despite the dedicated real-parameter EAs, such as evolution strategy, differential evolution etc., real-parameter GAs have gained adequate popularity in the recent past. The main challenge in developing an efficient real-parameter GA lies in devising a recombination operator in which two or more real-parameter vectors must be blended to create two or more offspring vectors of real numbers [11, 2, 9, 7]. These recombination operators employ a non-uniform probability distribution around the parent solutions to create an offspring solution. A theoretical study [1] attempted to find similarities among these operators.

In this paper, we concentrate on a particular recombination operator – the simulated binary crossover (SBX) operator [2]. The SBX operator uses two parent vectors and apply the blending operator variable by variable to create two offspring solutions. The operator involves a parameter, called the distribution index (η_c), which is kept fixed to a non-negative value throughout a simulation run. If a large value of η_c is chosen, the resulting offspring solutions are close to the parent solutions. On the other hand, for a small value of η_c, solutions away from parents are likely to be created. Thus, this parameter has a direct effect in controlling the spread of offspring solutions. Since a search process of finding the minimum solution of a function landscape largely depends on controlling the spread (or diversity) of offspring solutions vis-a-vis the selection pressure introduced by the chosen selection operation, fixing the η_c parameter to an appropriate value is an important task.

Here, we suggest a self-adaptive procedure of updating the η_c parameter by using the extension-contraction concept in a classical optimization algorithm. If the created child solution is better than the participating parent solutions, the child solution is extended further in the hope of creating even a better solution. On the other hand, if a worse solution is created, a contraction is performed. Either task will result in a update of η_c, so that the newly-created extended or contracted offspring solution has an identical probability of creation with an updated η_c'. This modification procedure has been applied to three single-objective and three two-objective optimization problems and compared with corresponding GAs with a fixed η_c value. In all cases, better performance of the suggested self-adaptive procedure is observed.

In the remainder of the paper, we briefly describe the SBX operator. Thereafter, we suggest the self-adaptive η_c update procedure and show simulation results on single-objective optimization problems. In each case, a parametric study with a parameter α is performed to find a suitable working range of this parameter. Then, a scale-up study by varying the number of decision variables is performed to demonstrate the polynomial complexity of the suggested algorithm. Finally, the self-adaptive update of η_c is extended to multi-

GECCO'07, July 7–11, 2007, London, England, United Kingdom.

objective optimization and results are discussed. Finally, conclusions from the study are made.

2. SIMULATED BINARY CROSSOVER (SBX)

As the name suggests, the SBX operator [2, 6] simulates the working principle of the single-point crossover operator on binary strings. In the above-mentioned studies, it was shown that this crossover operator *respects* the *interval* schemata processing [7], in the sense that common interval schemata between the parents are preserved in the offspring. 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:

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

First, a random number u_i between 0 and 1 is created. Thereafter, from a specified probability distribution function, the ordinate β_{q_i} is found so that the area under the probability curve from zero to β_{q_i} is equal to the chosen random number u_i. The probability distribution used to create an offspring is derived to have a similar *search power* to that in a single-point crossover in binary-coded GAs and is given as follows [2]:

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

Figure 1 shows the above 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 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 (thereby allowing a focussed search) and a small value of η_c allows distant solutions to be selected as offspring (thereby allowing to make diverse search).

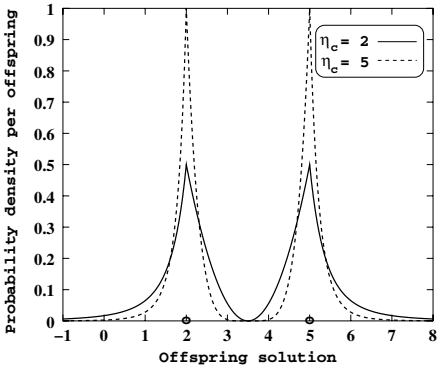

Figure 1: The probability density function for creating offspring under an SBX-η_c operator.

After obtaining β_{q_i} from the above probability distribution, the offspring are calculated as follows:

$$x_i^{(1,t+1)} = 0.5\left[(1+\beta_{q_i})x_i^{(1,t)} + (1-\beta_{q_i})x_i^{(2,t)}\right], \quad (3)$$

$$x_i^{(2,t+1)} = 0.5\left[(1-\beta_{q_i})x_i^{(1,t)} + (1+\beta_{q_i})x_i^{(2,t)}\right]. \quad (4)$$

The SBX operator biases solutions near each parent more favorably than solutions away from the parents. Essentially, the SBX operator has two properties:

1. The difference between the offspring is in proportion to the parent solutions.
2. Near-parent solutions are monotonically more likely to be chosen as offspring than solutions distant from parents.

An interesting aspect of this crossover operator is that for a fixed η_c the offspring have a spread which is proportional to that of the parent solutions

$$\left(x_i^{(2,t+1)} - x_i^{(1,t+1)}\right) = \beta_{q_i}\left(x_i^{(2,t)} - x_i^{(1,t)}\right). \quad (5)$$

In initial populations, where the solutions are randomly placed, making the difference in parents ($(x_i^{(2,t)} - x_i^{(1,t)})$) large, this allows to create a child solution which is also far away from the parents. However, when the solutions tend to converge due to the action of genetic operators (thereby making the parent difference small), distant solutions are not likely to occur, thereby focusing the search to a narrow region. As we discuss in the following section that this aspect of self-adaptive nature of SBX operator is not adequate alone in reaching near the optimum in large-sized and complex functions.

3. SELF-ADAPTIVE SBX

The SBX operator discussed above involves a parameter: the distribution index η_c. In most applications of SBX, a fixed value of $\eta_c = 2$ is used for single-objective optimization [2]. For a fixed value of η_c, the difference between the created child solution and the closest parent solution depends on the net difference between the two parent solutions, thereby causing a self-adaptive property of such a operator [5]. It has always been a research issue whether such a self-adaptive property is adequate in solving difficult optimization problems. Past studies of real-coded genetic algorithms with the SBX operator was not found to be suitable for multi-modal problems, such as Rastrigin's function [4]. Here, we suggest a procedure for updating the η_c parameter adaptively to solve such problems.

To illustrate the modified procedure, let us consider Figure 2, in which a typical child solution c is shown to be created from two parent solutions p_1 and p_2 by using SBX with a distribution index of η_c. Say, the random number used for this particular crossover operation is u ($\in (0,1)$). Then, if β (> 1) corresponds to the spread factor for the specific case of crossover, we obtain the following from the definition of spread factor:

$$\beta = 1 + \frac{2(c-p_2)}{p_2 - p_1}. \quad (6)$$

By noting that

$$\int_1^{\beta} \frac{0.5(\eta_c+1)}{\beta^{\eta_c+2}} d\beta = (u - 0.5),$$

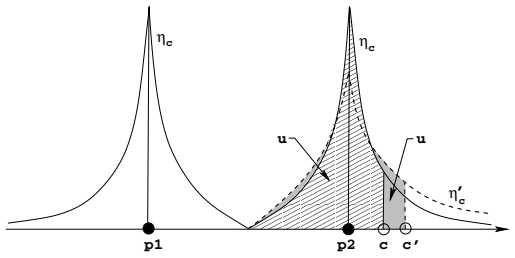

Figure 2: A schematic diagram showing the η_c update procedure.

we obtain the following relationship:

$$\eta_c = -\left(1 + \frac{\log 2(1-u)}{\log \beta}\right). \quad (7)$$

We now get into four scenarios (resulting in equations 9 to 12), as discussed below.

If this child solution c is better than both parent solutions, we assume that the region in which the child solution is created is better than the region in which the parents solutions are and intend to extend the child solution further away from the closest parent solution (p_2, in the case shown in the figure) by a factor α (> 1). This idea is similar to that followed in Nelder and Meade's simplex search method [10], in which an intermediate solution is either expanded or contracted or kept the same depending on the function value of the solution compared to that of the previously computed solutions. Rechenberg's 1/5th rule also keeps track of proportion of successful mutations over a certain number of trials [13]. If the success happens too often, the mutation strength is increased to create solutions away from the parent, else the mutation strength is reduced. By using $\alpha > 1$, the difference between the new child solution c' with closest parent p_2 is made α times than that between original child solution c and p_2, such that

$$(c' - p_2) = \alpha(c - p_2).$$

We then modify the distribution index to a value η_c' such that with this value and having the identical random number u, the modified child solution c' gets created from parent solutions p_1 and p_2, thereby yielding

$$\eta_c' = -\left(1 + \frac{\log 2(1-u)}{\log \beta'}\right). \quad (8)$$

Here, β' is the corresponding distribution index, given by

$$\begin{aligned} \beta' &= 1 + \frac{2(c' - p_2)}{(p_2 - p_1)}, \\ &= 1 + \alpha(\beta - 1). \end{aligned}$$

Using the above relationship between β' and β and using equations 7 and 8, we obtain the following update relationship for a child solution being found to be better than the nearest parent solution and the child lies outside the region bounded by parents:

$$\eta_c' = -1 + \frac{(\eta_c + 1)\log \beta}{\log(1 + \alpha(\beta - 1))}. \quad (9)$$

To make the η_c' value meaningful and free from numerical underflow error, we restrict it within $[0, 50]$, that is if $\eta_c' < 0$ is obtained by the above equation, we set it to be zero and if $\eta_c' > 50$ is used, we set it to be 50. This equation gives us an update procedure of η_c for providing a distribution index which is able to create children solutions in the right direction away from the parents. It is interesting to note that if a child c was created to the left of p_1, a similar update relationship will also be achieved.

If the child solution c is worse than both parent solutions, we would like to move the modified child c' to get closer to the parents and we may use $1/\alpha$ instead of α in equation 6:

$$\eta_c' = -1 + \frac{(\eta_c + 1)\log \beta}{\log(1 + (\beta - 1)/\alpha)}. \quad (10)$$

However, if a child solution is created inside the region bounded by both parents, a different update relationship will be obtained, since the SBX probability distribution function is different in this case. By following the above arguments, we obtain $\beta' = \beta^{\alpha}$ and the relationship for an improved child solution between η_c and η_c' values is obtained as follows:

$$\eta_c' = \frac{1 + \eta_c}{\alpha} - 1. \quad (11)$$

Once again, we restrict its value within $[0, 50]$ to avoid any computational error and to make η_c' meaningful.

For a scenario of creating a worse child, the above update relationship changes to

$$\eta_c' = \alpha(1 + \eta_c) - 1. \quad (12)$$

If the child solution c has a function value which is within the function value of the two parent solutions, we set $\eta_c' = \eta_c$.

To implement the update concept, we assign a η_c value within $[\eta_c^L, \eta_c^U]$ in the initial population. In all simulations here, we use $\eta_c^L = \eta_c^U = 2$. For a child solution, one random number is created and a corresponding β is computed. This β is used for all n variables and a child is created by variable-wise application of the SBX operator. This results in a line-SBX operator which produces a child along the line joining the two parent vectors. Thereafter, depending on the event of whether a better child (than both parents) or a worse child (than both parents) is created, η_c is updated and is assigned to the corresponding child solution. We assign a new η_c' value to each child solution separately.

It is interesting to note that if $\alpha = 1$ is used, all the above update of η_c procedure result in $\eta_c' = \eta_c$ and the above procedure is identical to the fixed η_c (original SBX) procedure.

3.1 Effect of mutation operator

In addition to the above modified SBX operator, a mutation operator can be used to perturb the created child c'. In such an event, there is an extra solution evaluation per child creation. Solution c gets evaluated during the crossover operator and the mutated version of the modified solution c' is evaluated. In our implementation, we add such extra function evaluations in the computation of performance of the modified procedure. However, if the mutation probability is so low that no variable gets mutated by the mutation operator, there is no extra evaluation recorded.

4. SIMULATION RESULTS

We now apply the modified approach to three different unconstrained functions which are popularly used in the GA

literature. Many studies in the literature on the chosen test problems use a population which is initialized symmetrically around the global minimum of the functions. When an algorithm with such a population uses recombination and mutation operators, which have a tendency to create solutions in the central region of the search space bounded by the population members, it finds an easier time converging to the global minimum. To avoid any such undue advantage from the operators, in all simulations here, the initial population is bounded in the range $x_i \in [10, 15]$ for all i, such that the global minimum is not bracketed in the initial population. This provides a stiff test to an algorithm to first get out of the region bounded by the initial population and then keep moving in the correct direction so as to reach the global minimum.

4.1 Sphere Function

The sphere function is the simplest of the three functions used in this study:

$$f(\mathbf{x}) = \sum_{i=1}^{n} x_i^2. \tag{13}$$

First, we employ the real-coded GA with the original fixed-η_c based SBX operator on the 30-variable sphere function with following parameter settings: population size = 150, $p_c = 0.9$, $\eta_c = 2$, $p_m = 1/30$, and $\eta_m = 50$. A run is terminated when a solution having a function value equal to 0.001 is found. Eleven runs are made from random initial populations. Figure 3 shows a typical variation of best population function value with the number of function evaluations. To investigate the effect of recombination operator alone, we make another set of runs with $p_m = 0$ and a typical performance is also shown in the figure. Both results indicate that the fixed-η_c based GA is unable to find the true optimum in any reasonable number of evaluations for a 30-variable sphere function. However, it is worthwhile to mention here that after a variable-wise creation of two offspring vectors, if the variables values are swapped randomly between the two offsprings (similar to a uniform crossover operator or the crossover operator in differential evolution [12]), a much quicker convergence with a fixed $\eta_c = 2$ can be obtained for variable-wise separable functions, like the sphere function [5]. This is because variables can independently reach near the true optimum in different population members and a uniform swapping of such solutions can create a complete solution having near optimal variables values. However, this uniform-like crossover operation may not work well in rotated and more complex problems. In this paper, we only concentrate on the blending part of the operation and investigate whether the fixed nature of η_c or the self-adaptive nature of η_c is more effective without having any crossover-like swapping effect.

Next, we employ our self-adaptive update procedure of η_c. All initial population members are initialized with $\eta_c = 2$ and then allowed to change based on our update procedure described above. All other parameters are the same as above and to investigate the effect of the self-adaptive recombination operator alone, we use $p_m = 0$ and reduce the crossover probability to $p_c = 0.7$. For the η_c update we use $\alpha = 1.50$. A typical variation of the best population function value is shown in Figure 3 and is found to have good converging property. Table 1 shows the best, median and worst number of function evaluations to reach a solution with $f = 0.001$ out of 11 runs. It is clear that compared to the fixed-η_c schemes, the self-adaptive scheme is able to steer the search towards the true minimum solution quickly and converge close to the minimum. It is clear that even with 300,000 function evaluations, the fixed η_c scheme is not able to find a near-optimum solution, whereas with a median of 184,050 function evaluations, a solution with three decimal places accuracy from the true optimum is found repeatedly in 11 runs.

4.1.1 Parametric Study

Figure 4 does a parametric study with α in the range $[1.05, 2.00]$. Recall that the parameter α signifies the extent of change in the offspring solution performed to recompute the η_c parameter for an identical probability event of creating the modified solution. Once again, 11 runs are performed for every case and the best, median and worst number of function evaluations are shown in the figure. It can be said that the effect of α is not significant. Although there is a degradation of the median performance with increasing α, the performance is best in the range $\alpha \in [1.05, 1.50]$. Thus, despite the introduction of a new parameter (α) for updating an existing parameter (η_c), the effect of the parameter α is not significant.

4.1.2 Scale-up Study

Finally, we perform a scalability study by varying the number of variables (n) from 20 to 200. In this study, we have used the following update of parameter due to the increase in number of variables: population size = $5n$. All other parameters are kept the same as before and we use $\alpha = 1.50$. Figure 5 shows that (i) the real-coded GA with self-adaptive recombination operator is able to find a solution close to the true minimum (within a tolerance of 0.001 in the function value) and (ii) the increase in number of function evaluations is polynomial to the increase in number of variables ($O(n^{2.21})$). The GA with the self-adaptive SBX operator suggested here does not exploit the variable separability of the objective functions. This is the reason why the proposed GA takes more number of evaluations than other approaches which favor the variable separability and unimodality of the sphere function [5, 4].

4.2 Rosenbrock Function

Next, we consider the Rosenbrock's function:

$$f(\mathbf{x}) = \sum_{i=1}^{n-1} 100(x_i^2 - x_{i+1}) + (x_i - 1)^2. \tag{14}$$

This function has the global minimum at $x_i = 1$ for all i with a function value equal to zero. Near the minimum region, this function has a very small slope towards the minimum. This property of the landscape causes an algorithm to have a slow convergence to the minimum.

Once again, we first employ the original SBX operator with a fixed $\eta_c = 2$ and initialize $x_i \in [10, 15]$ for all i. Figure 6 shows a typical variation of population-best function value with the number of function evaluations for the 30-variable Rosenbrock function with following parameter settings: population size = 150, $p_c = 0.9$, $p_m = 1/30$, and $\eta_m = 50$. The algorithm is terminated when a solution with a minimum function value of 0.001 is found. The procedure is not able to come close to the true minimum. When we use

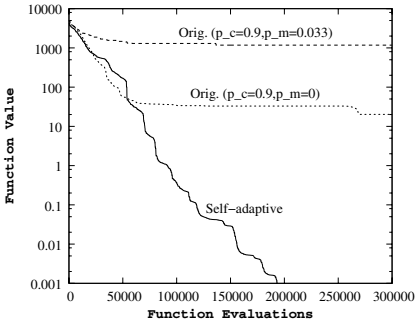

Figure 3: Variation of population-best function value with number of function evaluations for the 30-variable sphere function.

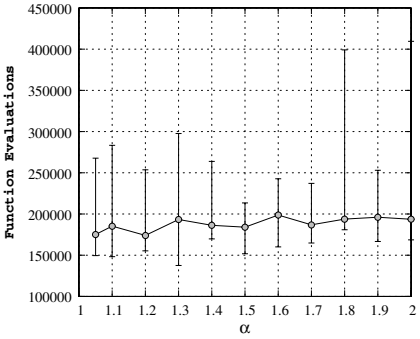

Figure 4: Parametric study of α for the 30-variable sphere function.

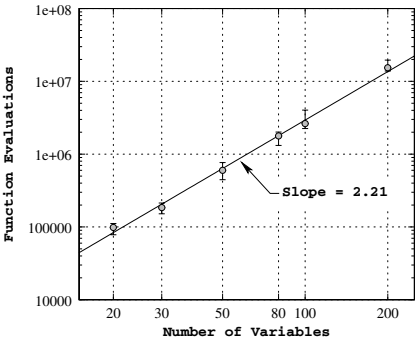

Figure 5: Scale-up study with number of variables for the sphere function.

Table 1: Performance of real-coded GAs with fixed and self-adaptive η_c update on 30-variable sphere function.

Method	Optimized function value (func. eval.)		
Original ($p_c = 0.9$, $p_m = 0.033$)	1.24e03 (300,000)	4.16e03 (300,000)	4.30e03 (300,000)
Original ($p_c = 0.9$, $p_m = 0$)	1.21e03 (300,000)	4.13e03 (300,000)	4.24e03 (300,000)
Self-adp.($p_c = 0.7$, $p_m = 0$)	10^{-3} (151,800)	10^{-3} (184,050)	10^{-3} (213,450)

$p_m = 0$, a somewhat better performance is observed, but the procedure is unable to reach near to the global minimum.

Now, we apply the self-adaptive recombination operator starting with $\eta_c = 2$ to all initial population members. We use $\alpha = 1.5$ for the update procedure. Figure 6 shows a typical variation in the population-best function value. The self-adaptive property of the recombination operator is able to adjust the η_c adequately to navigate through the fitness landscape to reach near the global minimum. Table 2 shows the best, median and worst performance out of 11 runs of original and self-adaptive GAs. For the self-adaptive case, we have shown results with $\alpha = 1.4$, which produces the best result. With as many as 10 million function evaluations, the fixed η_c scheme is not able to find a near-optimum solution (in fact, the best solution has a function value of $6.85(10^6)$), whereas with a median of about 6.9 million function evaluations, a solution with three decimal places accuracy from the true optimum is found repeatedly.

4.2.1 Parametric Study

A parametric study on α is shown in Figure 7. Although there is a upward trend in number of function evaluations with increasing α, with $\alpha \in [1.05, 1.50]$ the performance is better. Interestingly, in this problem also we find that the effect of α in a good range of values is not significant.

4.2.2 Scale-up Study

A scale-up study is made next by varying the number of variables from 20 to 200. Figure 8 shows that the number of function evaluations needed to reach up to three decimal places of accuracy varies as $O(n^{3.496}(\log n)^{-9.147})$ by the proposed self-adaptive procedure.

4.3 Rastrigin's Function

Next, we consider the Rastrigin's function which has many local optima and one global minimum at $x_i = 0$ for $i = 1, 2, \ldots, n$:

$$f(\mathbf{x}) = \sum_{i=1}^{n} x_i^2 + 10\,(1 - \cos(2\pi x_i))\,. \qquad (15)$$

In an earlier study [4], this function was difficult to solve for global optimality using the real-coded GA with a parent centric recombination operator, particularly when the initial population did not bracket the global optimum. In this study, we initialize a population with each x_i created randomly in $[10, 15]$, away from the global minimum solution. In a single dimension, an algorithm has to overcome at least 10 different local optima to reach to the global minimum. With a larger variable size, exponentially more local optima must be overcome to reach the global minimum.

First, we apply the real-coded GA with the original SBX operator on the 20-variable Rastrigin function with a standard parameter setting: population size = 100, $p_c = 0.9$, $\eta_c = 2$, $p_m = 1/20$, $\eta_m = 50$. GAs are run from 11 different initial populations till a maximum of 40,000 generations or till a solution having a function value of 10^{-4} is obtained. Table 3 shows the best function values obtained by the procedure. In this case, the best obtained function value (in 11 runs) with four million function evaluations is 319.523, whereas the globally best solution has a function value equal to zero. It is clear that no run is able to find a solution close to the global minimum. With $p_c = 0.7$ and $p_m = 0.01$, we obtain slightly better performance, but even now no solution close to the global minimum is found.

Next, we apply the real-coded GA with our proposed self-adaptive SBX operator with $p_m = 0.7$ and $p_m = 0.01$. All initial population members are initialized with $\eta_c = 2$ and then allowed to change using the proposed η_c update procedure described earlier. We use $\alpha = 1.5$ here. Table 3 shows

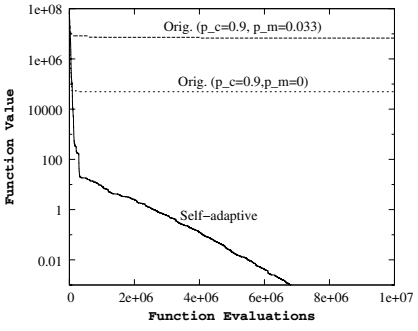

Figure 6: Variation of population-best function value with number of function evaluations for the 30-variable Rosenbrock's function.

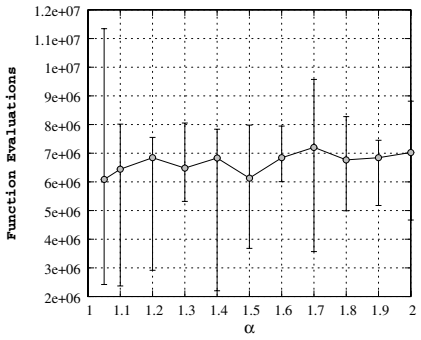

Figure 7: Parametric study of α for the 30-variable Rosenbrock's function.

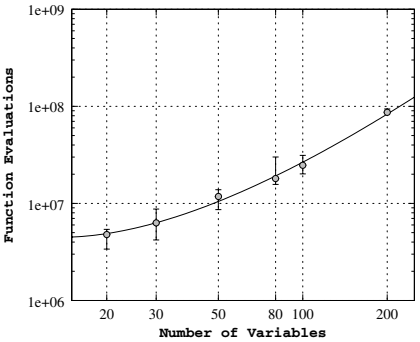

Figure 8: Scale-up study with number of variables for the Rosenbrock's function.

Table 2: Performance of real-coded GAs with fixed and self-adaptive η_c update on 30-variable Rosenbrock's function.

Method	Optimized function value (func. eval.)		
Original ($p_c = 0.9$, $p_m = 0.033$)	6.85e06 (10M)	7.44e06 (10M)	8.30e07 (10M)
Original ($p_c = 0.9$, $p_m = 0$)	9.94e06 (10M)	4.94e07 (10M)	5.19e07 (10M)
Self-adp.($p_c = 0.7$, $p_m = 0$)	10^{-3} (2,200,650)	10^{-3} (6,832,950)	10^{-3} (7,836,300)

that in all cases the self-adaptive update of η_c is able to find a solution with desired accuracy in a fraction of total evaluations used in the case of fixed-η_c procedures. Despite being started far away from the global minimum, the procedure is able to converge to the correct globally minimum solution. Figure 9 shows the decrease in best function value with number of function evaluations for a typical simulation run with a fixed $\eta_c = 2$ procedure ($p_c = 0.7$, $p_m = 0.01$) and with the self-adaptive procedure. The figure clearly shows that the fixed η_c run is poor in its performance, whereas the self-adaptive procedure steadily finds better and better solutions with function evaluations.

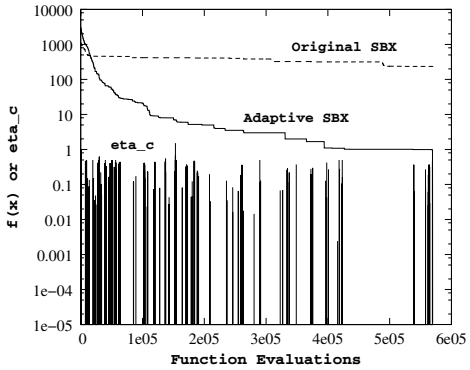

Figure 9: Decrease in best population function value with number of function evaluations for the 20-variable Rastrigin's function.

To understand the effect of the proposed self-adaptive update of η_c, we record the η_c value of the top 15 percentile solutions in terms of their function values and show its variation with number of function evaluations in Figure 9. The y-axis of this figure is made logarithmic. Although, the above η_c value seems to end at 10^{-5} in the figure, the actual value recorded by the procedure is $\eta_c = 0$. An interesting aspect is that the η_c value seems to take a value zero at various stages of the simulation. The line corresponding to the best function value (marked as 'Adaptive SBX') indicates that there are a number of function values, especially within $f = 10$ to $f = 1$, in which the algorithm seems to get stuck for a large number of evaluations before finding a better solution. Within this range of function values, this function has one local minimum at every integer value of the function, thereby having a possibility to get stuck 10 times. Interestingly, every time the best population member gets stuck at a local minimum, the η_c of the best 15 percentile solution gets updated to zero, thereby increasing the spread of created solutions by the SBX operator. Since an $\eta_c = 0$ will be the smallest possible η_c which provides the maximum spread in created solutions, the algorithm finds that the best way to counteract a local statis is to increase the diversity of created solutions to the extent possible. However, as soon as a better solution is found, the η_c is immediately updated to a value close to one, thereby providing a more focussed search around population members. With an assigned fixed value of η_c over the entire simulation run, such a variation in spread in solutions in an offspring population is not possible. The proposed procedure seems to employ this principle adaptively and multiply in as many times as the algorithms get stuck to a locally optimal solution and improve from such a situation. In the absence of such an adaptive update of η_c with the original SBX operator, the corresponding GA was not able to improve its performance efficiently every time it gets stuck to a locally optimal solution.

Table 3: Performance of real-coded GAs with fixed and self-adaptive η_c update on 20-variable Rastrigin's function.

Method	Optimized function value (func. eval.)		
Original ($p_c = 0.9$, $p_m = 0.05$)	319.523 (4M)	338.093 (4M)	342.363 (4M)
Original ($p_c = 0.7$, $p_m = 0.01$)	124.368 (4M)	210.929 (4M)	335.380 (4M)
Self-adp.($p_c = 0.7$, $p_m = 0.01$)	10^{-4} (287,822)	10^{-4} (429,511)	10^{-4} (569,597)

4.3.1 Parametric Study

A parametric study of α to investigate its effect on the performance of the proposed procedure is made next. For the 20-variable Rastrigin's function and with above parameter settings, 11 different runs are made. Figure 10 shows number of function evaluations needed to find a solution with a function value equal to or smaller than 10^{-4}. It is interesting to note that for a large range of values of α ($\in [1.05, 8.00]$), the performance of the proposed procedure remains fairly independent of α. However, the best performance seems to happen for $\alpha = 3$ with best, median and worst number of function evaluations of 221,082, 342,741, and 718,680, respectively.

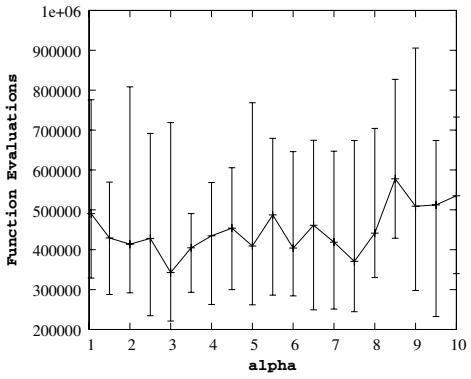

Figure 10: Effect of parameter α on number of function evaluations to obtain the global minimum with four decimal places of accuracy for the 20-variable Rastrigin's function.

4.3.2 Scale-up Study

Motivated by the success of the self-adaptive SBX procedure on the 20-variable Rastrigin's function, we now try to solve larger size Rastrigin's function with n varying in $[20, 200]$. With an increase in n, the number of local minima in a particular range of the variable values increase exponentially and the resulting problem is likely to provide more difficulty to an optimization algorithm. Since a function with a larger number of variables should ideally require a larger population size for a GA initialized with a randomly created population [8], we use the following parametric update for different n: population size=$5n$, $p_m = 0.2/n$, and $p_c = 0.7$. Here also, we initialize population with $x_i \in [10, 15]$, so as to not bracket the global minimum in the initial population. We also use $\alpha = 1.5$ for all n. We run till a solution with a function value equal to or smaller than 10^{-4} is obtained. Figure 11 shows that the required number of function evaluations increased polynomially ($O(n^{1.807})$) with n over the entire range of number of variables used in this study. It is

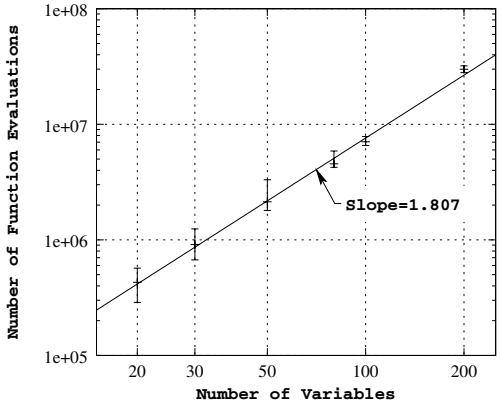

Figure 11: A polynomial increase in function evaluations with number of decision variables for the Rastrigin's function using proposed algorithm.

noteworthy that the proposed procedure is able to overcome exponentially many local minima to converge to the globally minimum solution with polynomially increasing number of function evaluations to as complex as a 200-variable Rastrigin's function.

5. SELF-ADAPTIVE SBX FOR MULTI-OBJECTIVE OPTIMIZATION

Like the way we made the SBX operator self-adaptive for single-objective optimization, we extend the idea here for multi-objective optimization. The main difficulty arises in deciding when a child solution is better than a parent. Here, we simply use the idea of non-domination to decide on this matter. Let us consider Figure 12. For the two parent objective vectors shown in the figure, if a child lies on the non-dominated shaded region (marked with 'A'), we call that the child to be better than the parents and we use the η_c update equations described earlier. On the other hand, if the child lies on the region marked as 'B' in the figure, we call that the child is worse than the parents and we use the appropriate equation (described earlier) to update η_c. If, however, the child lies on the region marked as 'C' in the figure, we do not update η_c. The remaining part of the NSGA-II algorithm [3] is used as usual.

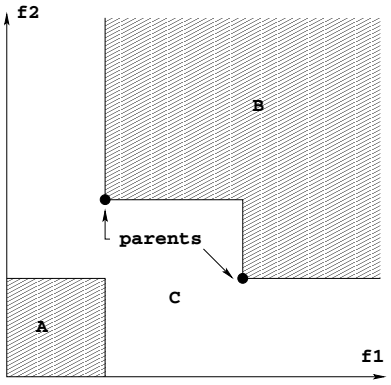

Figure 12: A sketch showing different regions in which the child may lie for an appropriate update of η_c.

6. SIMULATION RESULTS

Here, we show the working of NSGA-II with the self-adaptive SBX on three test problems: 30-variable ZDT1 and ZDT2 and 12-variable DTLZ2 problems.

6.1 ZDT1, ZDT2 and DTLZ2 Problems

We use the hyper-volume measure to indicate the performance of a procedure. We compare the performance of the self-adaptive procedure with NSGA-II having the fixed-η_c based SBX procedure with standard setting: $\eta_c = 15$ and $\eta_m = 20$. We use the following parameter setting for self-adaptive EA: population size = 100, $p_c = 0.9$, $p_m = 1/30$, $p_m = 0$ and maximum number of generations = 500. We initialize all population members with $\eta_c = 2$, as before. Table 4 shows the hyper-volume measure for various α values with the self-adaptive procedure. We observe that with α values near 1.70, the performance of the self-adaptive NSGA-II is better than the fixed-η_c based NSGA-II. Table 4 also shows the performance for 30-variable ZDT2 problem. Here, we observe that for α values larger than 1.20 the performance of the self-adaptive NSGA-II is better.

For the three-objective DTLZ2 problem, self-adaptive EA obtained better hyper-volume values of 0.57461, 0.56975 and 0.56597, compared to 0.55993, 0.55645 and 0.55539 as best, median and worst values obtained using original fixed-η_c EA.

Table 4: Performance (hyper-volume) comparison of self-adaptive NSGA-II with fixed-η_c based SBX on ZDT1 and ZDT2.

α	Best	Median	Worst
ZDT1			
Original, fixed $\eta_c = 15$			
	0.72745	0.72133	0.72075
Self-Adaptive SBX			
1.05	0.72768	0.68531	0.62985
1.20	0.73949	0.72203	0.56250
1.50	0.72908	0.72222	0.61360
1.70	0.73984	0.72228	0.72210
2.00	0.72282	0.72238	0.72196
ZDT2			
Original, fixed $\eta_c = 15$			
	0.38883	0.38846	0.38800
Self-Adaptive SBX			
1.05	0.38959	0.37661	0.15350
1.20	0.38940	0.38912	0.38874
1.50	0.38957	0.38920	0.38891
1.70	0.39040	0.38930	0.38890
2.00	0.38942	0.38922	0.38880

7. CONCLUSIONS

In this paper, we have suggested a self-adaptive procedure for updating the distribution index η_c used in the simulated binary crossover or SBX operator which is a commonly-used real-parameter recombination operator. The update procedure mimics the extension-contraction concept in Nelder and Meade's simplex search procedure and also follows, in principle, Rechenberg's 1/5-th update rule. On three single-objective optimization problems and on three two-objective optimization problems, the suggested procedure is found to be perform much better than the original SBX procedure. Further investigations are now needed for solving problems having a *linkage* among variables and problems having more than two objectives. A similar self-adaptive idea can also be used with other real-parameter recombination operators.

8. ACKNOWLEDGMENT

This study is supported by a research grant from Honda R&D, Japan.

9. REFERENCES

[1] H.-G. Beyer and K. Deb. On the desired behavior of self-adaptive evolutionary algorithms. In *Parallel Problem Solving from Nature VI (PPSN-VI)*, pages 59–68, 2000.

[2] K. Deb and R. B. Agrawal. Simulated binary crossover for continuous search space. *Complex Systems*, 9(2):115–148, 1995.

[3] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan. A fast and elitist multi-objective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002.

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

[5] K. Deb and H.-G. Beyer. Self-adaptation in real-parameter genetic algorithms with simulated binary crossover. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-99)*, pages 172–179, 1999.

[6] K. Deb and A. Kumar. Real-coded genetic algorithms with simulated binary crossover: Studies on multi-modal and multi-objective problems. *Complex Systems*, 9(6):431–454, 1995.

[7] L. J. Eshelman and J. D. Schaffer. Real-coded genetic algorithms and interval-schemata. In *Foundations of Genetic Algorithms 2 (FOGA-2)*, pages 187–202, 1993.

[8] D. E. Goldberg, K. Deb, and J. H. Clark. Genetic algorithms, noise, and the sizing of populations. *Complex Systems*, 6(4):333–362, 1992.

[9] T. Higuchi, S. Tsutsui, and M. Yamamura. Theoretical analysis of simplex crossover for real-coded genetic algorithms. In *Parallel Problem Solving from Nature (PPSN-VI)*, pages 365–374, 2000.

[10] J. A. Nelder and R. Mead. A simplex method for function minimization. *Computer Journal*, 7:308–313, 1965.

[11] I. Ono and S. Kobayashi. 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)*, pages 246–253, 1997.

[12] K. V. Price, R. Storn, and J. Lampinen. *Differential Evolution: A Practical Approach to Global Optimization*. Springer-Verlag, Berlin, 2005.

[13] I. Rechenberg. *Evolutionsstrategie: Optimierung Technischer Systeme nach Prinzipien der Biologischen Evolution*. Stuttgart: Frommann-Holzboog Verlag, 1973.