1997

Selective Crossover: Towards Fitter Offspring

Chilukuri K. Mohan
Syracuse University, ckmohan@syr.edu

Selective Crossover: Towards Fitter Offspring

Chilukuri K. Mohan

2-120 Center for Science and Technology,

Dept. of Electrical Engg. and Computer Science,

Syracuse University

Syracuse, NY 13244-4100, U.S.A.

ckmohan@syr.edu, 315-443-2322/(fax) 1122

August 19, 1997

Selective Crossover: Towards Fitter Offspring

Abstract: A new general-purpose crossover operator is proposed. The representation of a candidate solution is slightly perturbed, and the ensuing changes in fitness are calculated. Such fitness changes (for parents) are used in constructing the offspring resulting from crossover. Experiments with several sets of problems demonstrate that this approach leads to rapid increases in average and best fitness, and performs much better than traditional general-purpose crossover operators.

1 Why another crossover operator?

Actress to G.B.Shaw: Let's get married; our children will be as good-looking as me and as intelligent as you.

G.B.Shaw: Sorry, I am afraid the opposite may happen.

Actress: Fear not, we'll use Selective Crossover.

The "science" in most search techniques expounds on general-purpose weak mechanisms that can be readily applied to any problem, but are inefficient because their generality blinds them to the idiosyncrasies of each specific problem. The "technology," on the other hand, applies the approach using special-purpose mechanisms carefully tailored to work well on specific problems, but of little interest outside that limited domain. In the context of evolutionary algorithms, this dichotomy occurs in the choice of the crossover operators: well-known general-purpose operators such as one-point crossover [5] can be contrasted with specialized operators such as partially mapped crossover [3] for sequencing problems. The challenge is to develop operators of general applicability, that can also exploit problem-specific characteristics: this suggests that the results of applying operators depend in some way upon the fitness function.

In most evolutionary algorithms, the selection process is completely separated from the offspring-generation process, and only the former depends on fitness. This traditional approach ignores the fact that useful fitness-related information can be extracted even from individuals of low fitness. An offspring can be engineered to inherit those genes from each parent that are likely to improve fitness, on the basis of observation of the fitness of the parent, if we are allowed the luxury of evaluating the fitness of other hypothetical individuals that differ from the parent in specified ways. The information available in this manner from a population of diverse individuals (of which many are of low fitness) is much more valuable than a less-diverse population with higher fitness individuals. Indeed, convergence of a population may be a byproduct of evolution, but is not necessarily desirable from the viewpoint of information preservation.

This paper presents a new general-purpose crossover operator that uses fitness information about the immediate "neighbors" of parent individuals. Where parents differ, the value of an offspring gene is chosen by evaluating the possible effect of changing that allele in each parent. Thus, even low-fitness individuals can help in producing high-fitness offspring, assuming sufficient population diversity. This can be contrasted with mechanisms that ignore low-fitness individuals and quickly fill up the population with near-clones from which further improvement becomes almost impossible.

Our present focus is on genetic algorithms that operate on binary strings. We demonstrate the superior performance of the new crossover operator on several classes of problems, comparing it with one-point, two-point, and uniform crossover.

Section 2 describes the new crossover operator. Experimental results are given in Section 3. Results are summarized in Section 4.

2 Selective Crossover

Figure 1 illustrates one version of the paraboloid approximation method (attributed to Newton), for iteratively minimizing a function in a single variable x. Given two points on the function, and its derivatives at those points, a new candidate solution is constructed by analytically finding the value of x that minimizes the parabola passing through those points (with the appropriate derivatives). Even if the original cost function is not paraboloid, there are certain conditions under which its true minimum is found quickly by repeated iterations of this method.

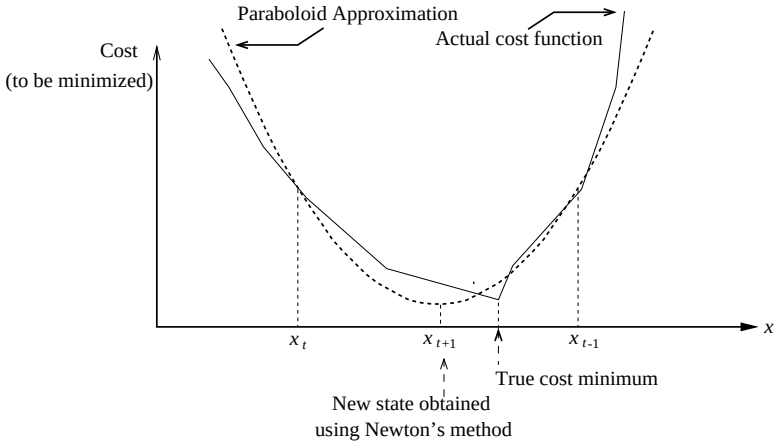

Figure 1: Approximating the function $Cost(x)$ (to be minimized) by a paraboloid

The above process of generating a new candidate solution from existing ones is analogous to crossover and recombination operators in evolutionary algorithms. For evolutionary algorithms applied to continuous optimization problems, we may use this procedure directly to determine new offspring; the paraboloid computation is applied to two members of the current population chosen to be parents for a recombination step. Our present focus is on discrete (bit-string) optimization problems solved by genetic algorithms; we merely borrow the spirit of this approach to define the new crossover operator. Instead of the gradient function from the paraboloid approximation method, we rely on the fitness difference when a small perturbation is made in an individual.

Let f be the fitness function to be maximized, and N the length of each bit-string (chromosome). For any j, $1 \leq j \leq N$, let $\boldsymbol{x}[\overline{j}]$denote the result of reversing the jth bit in $\boldsymbol{x}$, i.e., $(x[\overline{j}])_j = 1 - x_j$, and $(x[\overline{j}])_i = x_i$ for $i \neq j$.

• Selective Crossover (**SX**) of $\boldsymbol{x}$ and $\boldsymbol{y}$ yields $\boldsymbol{z} = [z_1, \ldots, z_N]$, where, for $1 \leq j \leq N$,

$$z_j = \begin{cases} x_j \text{ if } f(\boldsymbol{x}[\overline{j}]) - f(\boldsymbol{x}) \ < \ f(\boldsymbol{y}[\overline{j}]) - f(\boldsymbol{y}) \\ y_j \text{ otherwise.} \end{cases}$$

In other words, the jth bit of the offspring is chosen from that parent in which changing the jth bit causes less improvement (or more harm) than in the other parent.

Notes

- As in other traditional crossover operators, offspring are members of schema common to both parents; if $x_i = y_i$, then $z_i = x_i$. Perhaps breaking this constraint may also help the evolutionary process, an issue not addressed in this paper.

- In preliminary experiments, a more elaborate operator closer to the paraboloidal method also performed better than traditional crossover operators, but worse than Selective Crossover.

- To avoid recomputation, the fitness $f(\boldsymbol{x})$ and fitness differences $f(\boldsymbol{x}[\overline{j}]) - f(\boldsymbol{x})$ can be stored for each individual $\boldsymbol{x}$ (when these are first computed), and reused when needed. This is especially useful in elitist evolutionary algorithms in which fitter individuals tend to persist in the population for many generations.

- For many problems, the decision whether $f(\boldsymbol{x}[\overline{j}]) - f(\boldsymbol{x}) \ < \ f(\boldsymbol{y}[\overline{j}]) - f(\boldsymbol{y})$ can be made efficiently without actually computing the fitness of the perturbed individual $f(\boldsymbol{x}[\overline{j}])$ from scratch. For instance, in graph bipartitioning, this decision can be made by examining the sum of the weights on edges attached to nodes which are moved from one group to another in changing from $\boldsymbol{x}$ to $\boldsymbol{x}[\overline{j}]$.

3 Experimental Results

Three sets of bit-string function optimization experiments were conducted: a bimodal problem, concatenations of order-3 deceptive problems, concatenations of bipolar multimodal deceptive problems, and graph bipartitioning. For uniformity, the simple canonical generational genetic algorithm was used in each experiment, with roulette wheel reproduction selection and wholesale replacement of the entire population of parents by offspring, except for retaining the best solution (found so far) in the next generation. These are features of the genetic algorithm, and not of the crossover operators being compared.

All results reported are averages over ten trials. In each trial, the same data and (randomly generated) initial population were used for each crossover operator, and the only respect in which the algorithms differed was in the crossover operator. In each problem, the mutation rate was fixed to be roughly the reciprocal of the number of bits in the bit-strings (e.g., 0.01 for 100-bit representations). For the first two sets of experiments, each operator was applied to produce a single offspring in each crossover step; to check whether this unduly influences performance, in the third set of experiments (graph partitioning), two offspring were produced by each application of the traditional operators, and Selective Crossover was applied twice as often in each generation.

Computation times for each experiment were measured to check whether the new Selective Crossover operator involves substantial computational overhead. Numbers in Tables 1-4 (in Section 3) show that such is not the case, although computation times varied due to irrelevant implementation details, scheduling policy of the hardware (longer jobs receive lower priority), and machine load variations. Different machines [1] were used, but the same machine was used for each set of experiments.

[1] (1) Sun4/65 a.k.a. SPARCstation 1+ with sun4 cpu, 40MB main memory and 72MB virtual memory; (2) SPARCsystem 10 with sun4 cpu, 128MB main memory and 398 MB virtual memory; and (3) Sun Ultra 1, with sparc CPU, 156MB main memory and 686MB virtual memory.

3.1 A bimodal optimization problem

The following function is to be maximized:

$$f(\boldsymbol{x}) = MAX - (weight(\boldsymbol{x}) + 0.1) \times (distance(\boldsymbol{x}, SOL) + 0.01)$$

where MAX is large enough to ensure $f(\boldsymbol{x}) > 0$, $weight(\boldsymbol{x})$ is the number of 1's in $\boldsymbol{x}$, and $distance(\boldsymbol{x}, SOL)$ is the Hamming distance between $\boldsymbol{x}$ and a solution bit-string SOL. A local optimum occurs at the bit-string containing all zeroes, while the global optimum occurs at SOL.

Table 1: Performance of various operators on bimodal 30-bit problems, with pop. size =40

Performance criterion	One-Point	Two-Point	Uniform	Selective
Best fitness in 100 gens.	844.3	843.2	850.4	902.8
Best fitness in 1000 gens.	859.9	859.2	861.1	902.8
No. of gens. to achieve best	900	950	900	25
Time (sec./gen. on Sparc10)	0.18	0.19	0.24	0.35

For the bimodal problem, the results (reported in Table 1) were overwhelmingly in favor of the new operator. Note that the third row of entries in the table indicates the number of generations beyond which best fitness (averaged over 10 trials) ceased to improve until the genetic algorithm trials were terminated. For instance, best fitness of 859.9 was achieved using one-point crossover in $\leq$900 generations, and the next 100 generations yielded no further improvement.

The fourth row of entries in the table indicates the average time per generation, i.e., total run-time divided by the number of generations.

3.2 Concatenations of order-3 deceptive problems

Goldberg and associates [1] have conducted extensive studies on variants of the following "easy" and "ugly" problems obtained by concatenating n order-3 deceptive problems. If A is a bit-string, then

$$easy_{3n}(A) = \begin{cases} 0 \text{ if } n = 0 \\ easy_{3n-3}(A) + f_3(3n-2, 3n-1, 3n, A) \text{ otherwise} \end{cases}$$

$$ugly_{3n}(A) = \sum_{i=1}^{n} f_3(i, i+n, i+2n, A)$$

$$\text{where } f_3(i,j,k,A) = \begin{cases} 0.9 \text{ if } A[i]+A[j]+A[k] = 0 \\ 0.6 \text{ if } A[i]+A[j]+A[k] = 1 \\ 0.3 \text{ if } A[i]+A[j]+A[k] = 2 \\ 1.0 \text{ if } A[i]+A[j]+A[k] = 3 \end{cases}$$

Global optima (valued at n for $easy_{3n}$ and $ugly_{3n}$) occur at the bit-strings containing all 1's, but there are $2^n - 1$ other local optima. Results in Table 2 and Figure 2 show that the new operator performs better than the traditional ones, especially for the larger versions of the problem. The difference in performance was so large that there was no need to continue beyond 1000 generations with the new operator. Various traditional operators had roughly equal performance.

Table 2: Performance of various operators on concatenations of Goldberg's order-3 deceptive problems

Problem	Performance criterion	One-Point	Two-Point	Uniform	Selective
$easy_{30}$ (pop. size 30)	Best fitness in 100 gens.	8.76	8.9	8.7	9.98
	Best fitness in 1000 gens.	9.17	9.2	9.12	10.0
	No. of gens. to achieve best	875	900	725	125
	Time (sec./gen. on Sparc10)	0.086	0.089	0.126	0.145
$ugly_{30}$ (pop. size 30)	Best fitness in 100 gens.	8.78	8.76	8.79	9.94
	Best fitness in 1000 gens.	9.22	9.18	9.13	10.0
	No. of gens. to achieve best	925	950	650	250
	Time (sec./gen. on Sparc10)	0.102	0.093	0.134	0.178
$ugly_{300}$ (pop. size 40)	Best fitness in 100 gens.	66.24	64.72	65.86	98.46
	Best fitness in 1000 gens.	68.04	64.72	68.66	98.47
	No. of gens. to achieve best	1000	100	1000	600
	Time (sec./gen. on Sparc10)	0.605	0.600	0.942	1.129
$ugly_{300}$ (pop. size 300)	Best fitness in 100 gens.	67.47	68.49	68.89	100.0
	Best fitness in 1000 gens.	69.23	69.43	69.8	
	Best fitness in 10000 gens.	69.45	70.49	71.04	
	No. of gens. to achieve best	4500	8500	8750	100
	Time (sec./gen. on Ultra1)	0.42	0.50	0.80	0.93

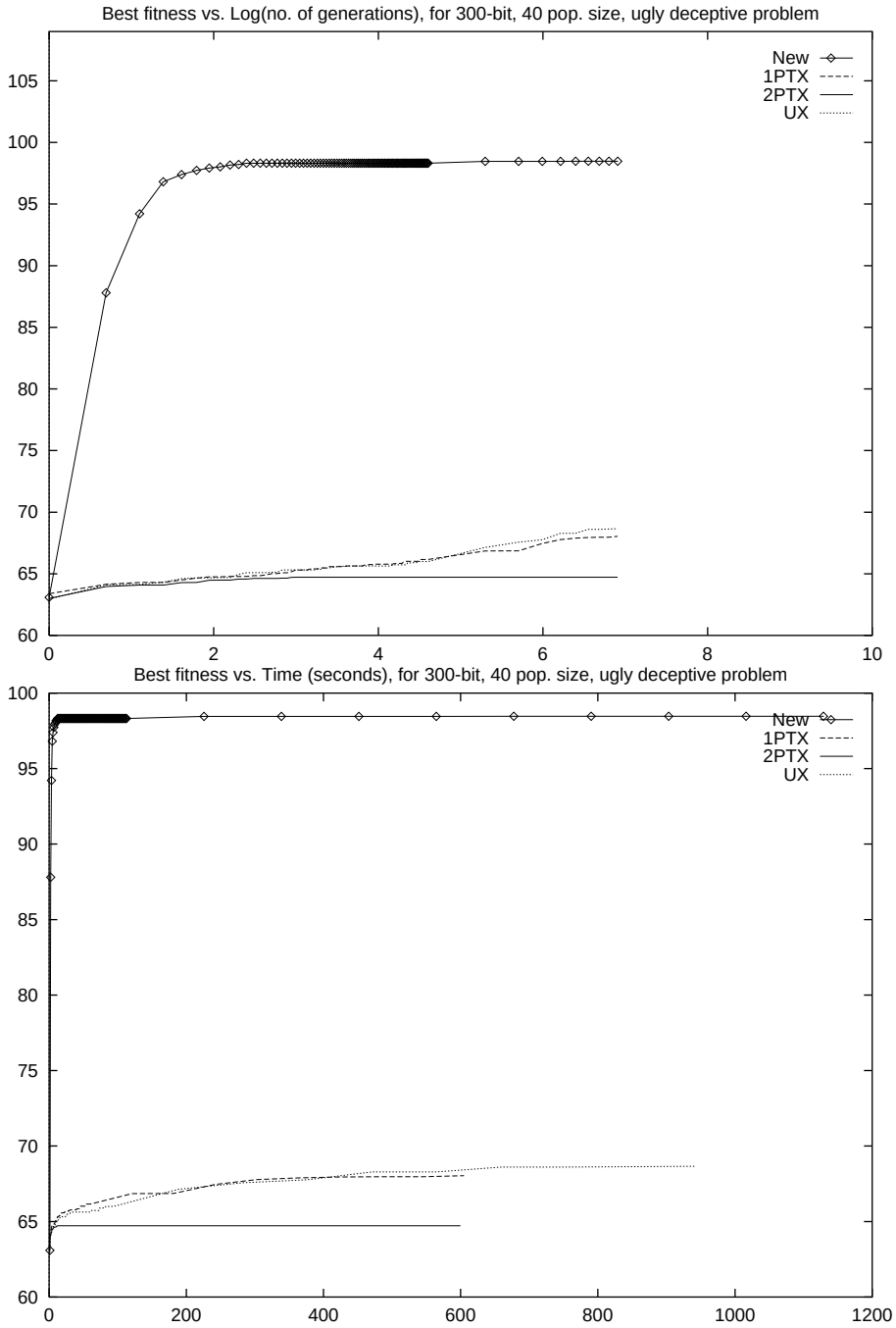

Figure 2: Comparison of different crossover operators for $ugly_{300}$, population size 40: evolution of best fitness with (a) log(number of generations), and (b) execution time (= number of generations $\times$ average time per generation)

3.3 Concatenations of bipolar deceptive problems

Goldberg *et al.* [4] define the class of *bipolar* deceptive functions with many more local optima than the order-3 deceptive functions described above. Results are reported below for the instance of this class of functions defined as follows. If A is a bit-string, then

$$bieasy_{6n}(A) = \begin{cases} 0 \text{ if } n = 0 \\ bieasy_{6n-6}(A) + b_6(6n-5, 6n-4, 6n-3, 6n-2, 6n-1, 6n, A) \text{ otherwise} \end{cases}$$

$$biugly_{6n}(A) = \sum_{i=1}^{n} b_6(i, i+n, i+2n, i+3n, i+4n, i+5n,\ A)$$

$$\text{where } b_6(i,j,k,l,m,n,\ A) = \begin{cases} 1.0 \text{ if } A[i]+A[j]+A[k]+A[l]+A[m]+A[n] = 0 \\ 0.0 \text{ if } A[i]+A[j]+A[k]+A[l]+A[m]+A[n] = 1 \\ 0.4 \text{ if } A[i]+A[j]+A[k]+A[l]+A[m]+A[n] = 2 \\ 0.8 \text{ if } A[i]+A[j]+A[k]+A[l]+A[m]+A[n] = 3 \\ 0.4 \text{ if } A[i]+A[j]+A[k]+A[l]+A[m]+A[n] = 4 \\ 0.0 \text{ if } A[i]+A[j]+A[k]+A[l]+A[m]+A[n] = 5 \\ 1.0 \text{ if } A[i]+A[j]+A[k]+A[l]+A[m]+A[n] = 6 \end{cases}$$

b_6 has two global optima and 20 deceptive local optima. Thus $bieasy_{30}$ has 32 global optima and over five million local optima, a problem that is not exactly trivial. For such a problem, Goldberg *et al.* [4] report that the GA is able to overcome deception and converge to a global optimum (for $easy_{30}$) if the population is large enough (> 391), and that convergence usually occurred in 50 generations.

Results using the new operator are presented in Table 3, for the genetic algorithm described just before Section 3.1. In one set of experiments, mutation was disabled, in order to examine the possibility whether any differences in results can be attributed to mutation. Numbers (in the fourth set of results in Table 3) show that mutation did not play a significant role in the success of the new operator; indeed, mutation was important for maintaining sufficient diversity in the population. In the absence of mutation, there was no improvement in the

population after the first 50 generations for any of the crossover operators, but this was not a problem for the new operator since it had already led to global optima.

Table 3: Performance of various operators on concatenations of bipolar deceptive problems

Problem	Performance criterion	One-Point	Two-Point	Uniform	Selective
$bieasy_{30}$ (pop. size 30)	Best fitness in 100 gens.	4.42	4.32	4.28	4.94
	Best fitness in 1000 gens.	4.54	4.50	4.52	5.0
	No. of gens. to achieve best	875	900	725	250
	Time (sec./gen. on Sparc10)	0.055	0.063	0.100	0.093
$biugly_{30}$ (pop. size 30)	Best fitness in 100 gens.	4.40	4.28	4.36	4.94
	Best fitness in 1000 gens.	4.50	4.48	4.46	5.0
	No. of gens. to achieve best	1000	850	900	150
	Time (sec./gen., on Sun4/65)	0.202	0.209	0.278	0.328
$bieasy_{30}$ (pop. size 400)	Best fitness in 100 gens.	4.60	4.50	4.60	5.0
	Best fitness in 1000 gens.	4.74	4.74	4.64	
	No. of gens. to achieve best	700	1000	425	5
	Time (sec./gen. on Sun4/65)	1.67	3.35	1.97	2.76
$bieasy_{30}$ (pop. size 400) without mutation	Best fitness in 100 gens.	4.50	4.40	4.40	5.0
	Best fitness in 1000 gens.	4.50	4.40	4.40	
	No. of gens. to achieve best	50	50	50	50
	Time (sec./gen. on Sparc10)	1.35	1.53	1.72	2.92
$bieasy_{300}$ (pop. size 300)	Best fitness in 100 gens.	33.02	33.24	32.46	49.96
	Best fitness in 1000 gens.	34.46	34.44	33.08	49.98
	No. of gens. to achieve best	950	875	900	975
	Time (sec./gen. on Ultra1)	3.32	2.38	5.79	4.78

3.4 Graph bipartitioning

Nodes of a graph (with randomly generated connection weights) are to be divided into two groups. The requirement that each group must contain the same number of nodes is treated as a hard constraint; if needed, randomly chosen alleles are modified to enforce this constraint. This ensures that search is restricted to the feasible search space, to avoid the pesky distractions caused when the population is allowed to contain infeasible individuals.

The cost measure to be minimized is the sum of connection weights between nodes in different groups. For fitness-proportionate reproduction with sufficient selection pressure in favor of better solutions, the fitness function $1/cost^2$ was used in all the GP experiments conducted.

For all experiments on instances of graph partitioning, the population size was chosen to be the same as the number of nodes in the graph.

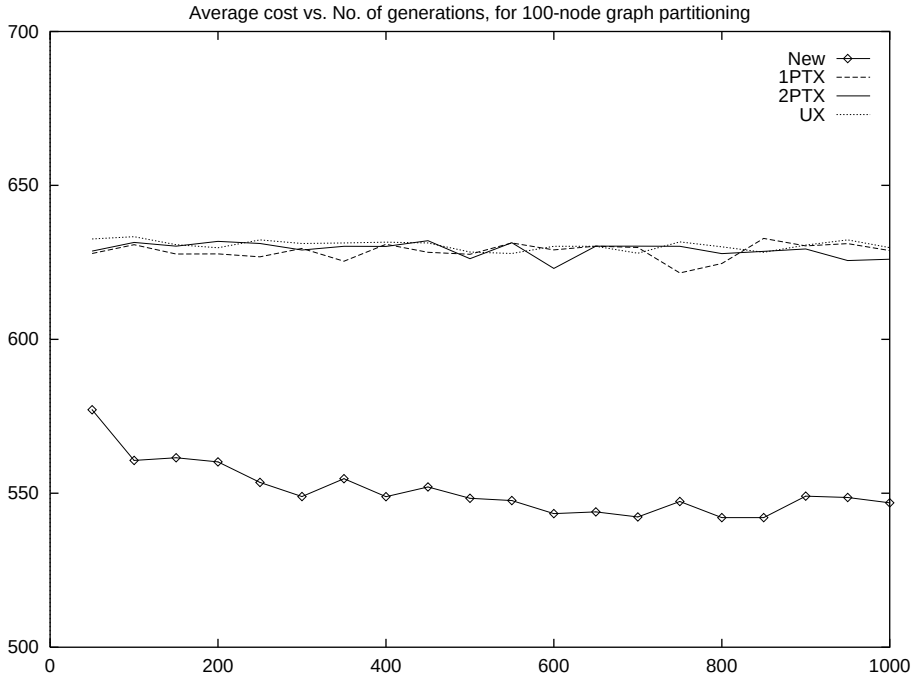

Figure 3: Evolution of average cost with number of generations, for 100 node graph bipartitioning problems, with population size 100

As shown in Table 4 and Figure 4, the new operator was significantly better than traditional operators for the larger instances (with 80 and 100 nodes) although the best cost for traditional operators was better for the smaller instances (with 30 and 40 nodes). In terms of average cost, the new operator always performed better than the traditional operators, as shown in Figure 3. The difference between average and best cost was substantial enough to believe that the population had not completely converged, i.e., all diversity had not been lost from the population.

Each application of the new operator is more expensive than the traditional ones, but the actual (measured) computation time required for a given number of generations is not much higher. Even for the large (100 node) instances of graph partitioning, 1000 generations were completed about in ten minutes. Figure 5 shows that the new operator required about half a minute to surpass the solution quality reached using one-point crossover in over 6 minutes.

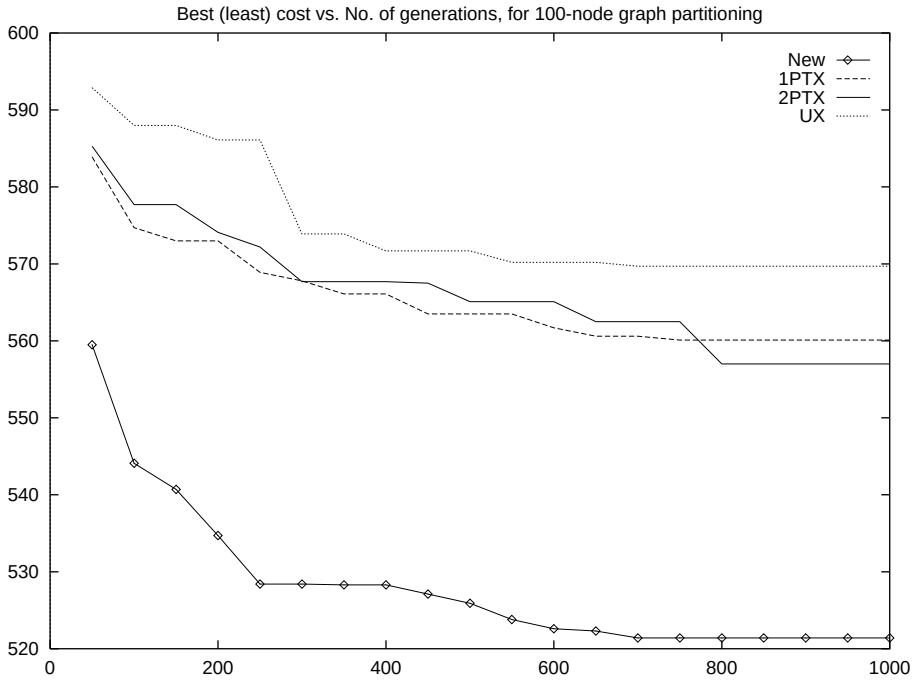

Figure 4: Evolution of best cost with number of generations, for 100 node graph bipartitioning problems, with population size 100

Table 4: Performance of various operators on graph bipartitioning problems.

#Nodes	Performance criterion	One-Point	Two-Point	Uniform	Selective
30	Best cost in 100 gens.	37.9	36.0	39.2	39.0
	Best cost in 200 gens.	35.9	**35.5**	37.5	38.6
	Avg. cost after 200 gens.	47.1	47.7	51.9	**45.5**
	#Gens. for best cost	140	160	170	180
	Time (sec./gen. on Ultra1)	0.032	0.055	0.035	0.043
40	Best cost in 100 gens.	74.7	80.3	78.5	77.6
	Best cost in 200 gens.	**72.8**	75.4	77.7	77.3
	Avg. cost after 200 gens.	93.6	94.9	94.5	**83.9**
	#Gens. for best cost	130	200	180	180
	Time (sec./gen. on Ultra1)	0.066	0.099	0.072	0.091
80	Best cost in 100 gens.	356.8	365.5	363.1	342.6
	Best cost in 1000 gens.	336.9	345.5	352.3	**328.5**
	Avg. cost after 1000 gens.	397.3	393.9	397.0	**347.8**
	#Gens. for best cost	850	800	600	500
	Time (sec./gen. on Ultra1)	0.252	0.655	0.640	0.294
100	Best cost in 100 gens.	574.7	588.0	577.7	544.1
	Best cost in 1000 gens.	560.1	557.0	569.7	**521.4**
	Avg. cost after 1000 gens.	628.9	626.0	629.8	**546.9**
	#Gens. to achieve best 1PTX results	750	800	-	50
	#Gens. in which best cost achieved	750	800	700	700
	Time (sec./gen. on Ultra1)	0.499	0.615	0.856	0.617

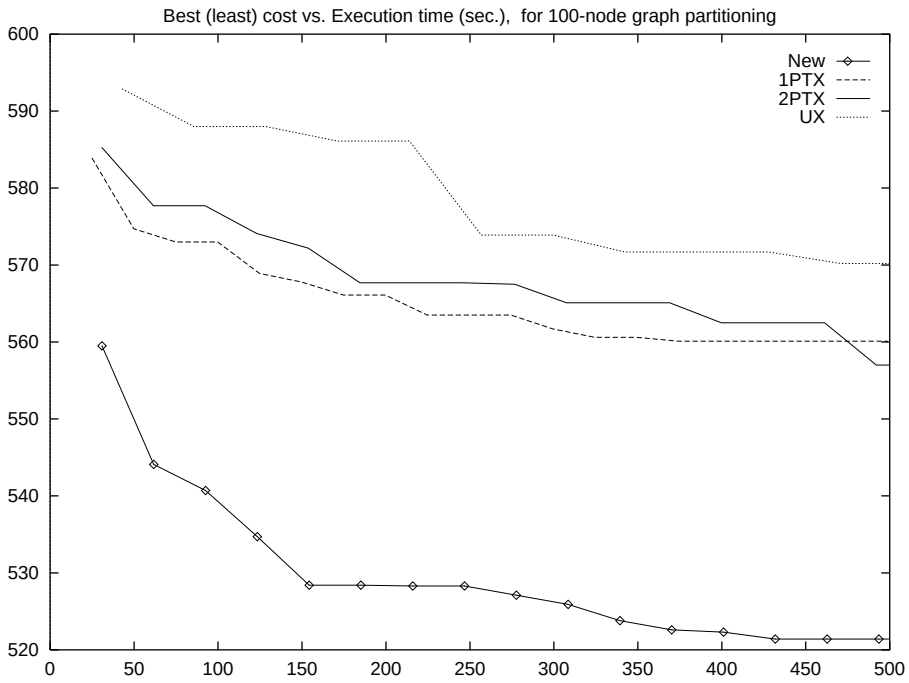

Figure 5: Evolution of best cost with time, for 100 node graph bipartitioning problems, with population size 100

4 Concluding Remarks

Selective Crossover is a new general-purpose operator for genetic algorithms, shown to outperform one-point, two-point, and uniform crossover on a bimodal problem, concatenations of order-3 deceptive problems, concatenations of bipolar multimodal deceptive problems, and large graph-bipartitioning problems. Although each application of the new operator requires many fitness evaluations than 1PTX, fewer generations are needed to obtain a specified solution quality, and much better results are obtained in a given amount of time by the new operator than by traditional crossover operators. Additional speedups can be obtained by storing and reusing 'gradient' information for individuals (when computed).

As expected with any "conservative" crossover operator in which every offspring allele is inherited from some parent, premature convergence may occur. Diversity may then be reintroduced into the population by temporarily increasing mutation rate, or by random

reinitialization of most of the population.

Since fitness-related information is being used in applying the new operator, it is conjectured that no reproduction selection mechanism is necessary. This conjecture is supported by experiments that yielded results of almost identical fitness even when fitness proportionate parent selection was disabled in the reproduction step (while using Selective Crossover). For instance, for the problem referred to as $ugly_{300}$, using random parent selection instead of fitness-proportionate selection (in the reproduction phase) lowered the best fitness obtained in 100 generations from 98.46 to 97.25 with population size 40, and from 100.0 to 99.98 with population size 300, averaging over 10 trials. For graph bipartitioning with 40 nodes and population size 40, omitting fitness-proportionate selection improved the best cost obtained in 100 generations from 77.6 to 77.3, averaging over 10 trials. These differences are small, suggesting the following 'heretic' algorithm:

Initialize population;

repeat

- Apply Selective Crossover to parents chosen randomly from current pop.;
- Replace entire population, except previous best, by the offspring generated;
- If convergence is detected, then re-initialize population, preserving the current best individual;

until desired solution fitness or computational limits are exceeded .

Researchers have recently explored the use of local improvement operators to improve the performance of evolutionary algorithms [7, 6]. Their emphasis is on steadily making small improvements to an individual; repeated fitness evaluations are performed, making transitions that lead to a local optimum. By contrast, the operator proposed in this paper uses local information **in** crossover, hoping to determine the better "building blocks" for the problem. A close analogy, in training neural networks, is the difference between "backpropagation"

[8] (using gradient descent) and the faster "quickprop" [2] algorithm (using the paraboloid approximation method).

References

[1] K. Deb and D. Goldberg, Analyzing Deception in Trap Functions, IlliGAL Tech. Report No. 91009, Univ. of Illinois at Urbana, 1991.

[2] S. E. Fahlman, An empirical study of learning speed in backpropagation networks, Technical Report CMU-CS-88-162, Carnegie-Mellon University, Pittsburgh (PA), 1988.

[3] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, Reading (MA), 1989.

[4] D. E. Goldberg, K. Deb, and J. Horn, *Massive multimodality, deception, and genetic algorithms*, Proc. Second Conf. Parallel Problem Solving from Nature, (eds. R. Manner and B. Manderick), Elsevier Science Pub., 1992.

[5] J. H. Holland, *Adaptation in Natural and Artificial Systems*, Univ. of Michigan Press, Ann Arbor (MI), 1975.

[6] P. Moscato, *On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts: Towards Memetic Algorithms*, Caltech Concurrent Comp. C3P Report 826, 1989.

[7] N. J. Radcliffe, and P. D. Surry, *Formal Memetic Algorithms*, Evolutionary Computing: AISB Workshop (Ed: T. Fogarty, Springer-Verlag), 1994.

[8] P. Werbos, *Beyond regression: New tools for prediction and analysis in the behavioral sciences*, Ph.D. thesis, Harvard University, Cambridge (MA), 1974.