01 Jul 1993

Genetic Algorithm with 3-parent Uniform Crossover

L. Vincent Edmondson

Billy E. Gillett
Missouri University of Science and Technology

Recommended Citation

Edmondson, L. Vincent and Gillett, Billy E., "Genetic Algorithm with 3-parent Uniform Crossover" (1993). *Computer Science Technical Reports*. 44.
https://scholarsmine.mst.edu/comsci_techreports/44

GENETIC ALGORITHM WITH 3-PARENT UNIFORM CROSSOVER

V. Edmondson* and B. Gillett

CSc-93-22

Department of Computer Science

University of Missouri - Rolla

Rolla, MO 65401 (314)341-4491

*This report is substantially the Ph.D. dissertation of the first author, completed July 1993.

This dissertation has been prepared using the publication thesis option. Pages 1-22 have been prepared in the style utilized by the journal *SIAM Journal on Computing* and will be presented for publication in that journal. Pages 23-47 have been prepared in the style utilized by the journal *ORSA Journal on Computing* and will be presented for publication in that journal. Pages 48-56 have been prepared in the style utilized by the journal *Missouri Journal of Mathematical Sciences*, where they appeared in the Winter 1993 issue.

ABSTRACT

A new genetic algorithm which uses a 3-parent uniform crossover operator is developed and analyzed. Uniform crossover operators are shown to be based on the premise that all bit-level genetic information should be passed from parents to children. The 3-parent uniform crossover operator is shown to adhere to this premise. The 3-parent uniform crossover operator is shown to be better than the 2-parent uniform crossover operator on the De Jong test functions.

Two new genetic algorithms which use 3-parent traditional crossover operators are developed and analyzed. The first uses a strategy of randomly selecting 3 of the 6 children resulting from 3-parent reproduction. The second uses a strategy of selecting the best 3 of the 6 children resulting from 3-parent reproduction. Each of the 3-parent traditional crossover operators is shown to be superior to the 2-parent traditional crossover operator on the De Jong test functions. The strategy of selecting the best 3 out of 6 children is shown to be superior to the strategy of randomly selecting 3 out of 6 children.

In addition to these 3-parent genetic algorithms, a relationship between the Metropolis algorithm from simulated annealing and the two-membered evolution strategy is developed. The Metropolis algorithm is shown to be a special case of the two-membered evolution strategy.

ACKNOWLEDGEMENTS

I would like to thank my advisor, Dr. Billy E. Gillett, and the members of my committee (past and present), Dr. Fikret Ercal, Dr. C. Y. Ho, Dr. John B. Prater, Dr. A. Kellam Rigler, and Dr. Sema Alptekin, for their help during my graduate work at UMR.

Thanks are also extended to UMR and the Computer Science Department for a Chancellor's Fellowship and graduate teaching assistantship.

I want to thank the members of the Department of Mathematics and Computer Science at Central Missouri State University for their support and patience. Special thanks go to Dr. Curtis Cooper and Dr. Robert Kennedy for continuing to show by example that excellence in teaching and excellence in research are not mutually exclusive.

I want to thank my in-laws, Jerry and Judy Murray, both for their encouragement and for providing me with a loving family structure.

Finally, I want to thank my wife Kelly for her continual love and support. The attainment of this degree would not have been possible without her.

TABLE OF CONTENTS

APPENDICES

LIST OF ILLUSTRATIONS

Figure Page

LIST OF TABLES

I. A GENETIC ALGORITHM WITH 3-PARENT UNIFORM CROSSOVER

A. ABSTRACT

A new genetic algorithm which uses a 3-parent uniform crossover operator is presented. The goal of the research was to obtain better results for the De Jong test functions using the 3-parent uniform crossover operator in comparison to the 2-parent uniform crossover operator. Uniform crossover operators are shown to be based on the premise that all bit-level genetic information should be passed from parents to children. The 3-parent uniform crossover operator is shown to adhere to this premise. The 3-parent uniform crossover operator is shown to be better than the 2-parent uniform crossover operator.

B. INTRODUCTION

Genetic algorithms (GAs) are randomized, population-based search procedures which utilize the Darwinian notion of survival of the fittest. These algorithms were developed independently by John Holland at the University of Michigan [1] and by Ingo Rechenberg and Hans-Paul Schwefel in Germany [2]. GAs have been applied in fields ranging from engineering and computer science to the social sciences [3]. It is anticipated that, because of their robust nature, GAs will continue to be applied to a wide variety of areas.

The traditional genetic algorithm (GA), as developed by Holland, begins with a population of randomly-generated binary string creatures. The fitness of each individual in the population is evaluated using an objective function and then these objective function values are used to determine which individuals will participate in the reproduction process. Selection for the reproduction process can be easily understood as a biased roulette wheel. Each individual is allocated an amount of the roulette wheel which is proportional to its objective function value. The actual reproduction process involves the two operators of crossover and mutation. The crossover operator exchanges bits (genetic information) between two parents. The mutation operator (which is invoked with only a small probability) is used to change a 0 to 1 or a 1 to 0. This perturbation is used to ensure that population diversity is maintained. This reproduction process is used to create a new generation of population members. The fitness of each individual in the new generation is then evaluated and the aforementioned process is repeated for either a preset number of generations or a preset amount of computer time.

C. UNIFORM CROSSOVER OPERATORS

The uniform crossover operator was primarily developed by David Ackley [4] and Gilbert Syswerda [5]. Each of the two most recent international conferences on GAs have included papers which focus on uniform crossover [6,7].

1. 2-parent Uniform Crossover. The 2-parent uniform crossover operator uses a crossover mask. This crossover mask is a string of bits in which the parity of each bit determines which parent will contribute the genetic information to the child. Each crossover mask has an inverse mask in which the parity of each bit in the crossover mask is reversed. For example, if a crossover mask is 01101, then its inverse mask is 10010.

The 0-bits and 1-bits in the 2-parent uniform crossover mask are uniformly distributed, occurring with probability 0.5 for each bit position. An algorithm for constructing a crossover mask and its inverse is given in Figure 1. Assume that the reference to the function *random (0,1)* will return either the digit 0 or the digit 1, each with probability 0.5.

```
let k = length of the bit-string
for j = 1 to k do
        mask[j] = random (0,1)
        inverse_mask[j] = (mask[j] + 1) MOD 2
```

Figure 1. 2-parent uniform crossover and inverse mask construction

The following theorem establishes a premise upon which the 2-parent uniform crossover operator is developed.

Theorem 1: If two children are produced from two parents using the 2-parent uniform crossover mask and its inverse, then all bit-level genetic information is maintained during the crossover portion of the reproduction process.

Proof: Let S_j represent the set resulting from the union of crossover and inverse mask values for a given bit-position j. If the cardinality of S_j is 2 for every bit-position j, then no genetic information can be lost because each parent contributes a bit-value to a child. If the crossover mask has a value of 0 for any position j, then the inverse mask will have $(0 + 1)$ MOD $2 = 1$ in position j. If the crossover mask has a value of 1 in position j, then the inverse mask will have a value of $(1 + 1)$ MOD $2 = 0$ in position j. Therefore, regardless of the value in position j of the crossover mask, the cardinality of S_j is 2 and no bit-level genetic information can be lost. Q.E.D.

Here is an example of reproduction using the 2-parent uniform crossover operator.

Parent 0: 011101
Parent 1: 101010
Mask: 101100
Inverse Mask: 010011
Child 0: 111001
Child 1: 001110

It is assumed that the two masks are generated using the algorithm shown in Figure 1. As is typical for uniform crossover, the children are decidedly different than the parents. Enumeration of the bit-level values for the parents shows that there are seven 1-bits and five 0-bits. As expected from Theorem 1, enumeration of the bit-level values for the children shows seven 1-bits and five 0-bits.

The 2-parent uniform crossover operator, along with the mutation operator, is used in the reproduction process as described above. It has been shown by Syswerda to be more effective than either the 1-point or 2-point traditional crossover operator [5].

2. 3-parent Uniform Crossover.

The 3-parent uniform crossover operator is a new reproduction operator that is a generalization of the 2-parent uniform crossover operator. It uses a crossover mask with position values ranging from 0 to 2 (inclusive). Under the assumption that *n* parents should generate *n* children, the algorithm for generating the 3-parent uniform crossover mask and its "inverses" is given in Figure 2. The "inverses" are defined in such a way that all bit-level genetic information is maintained throughout the crossover portion of the reproduction process. Assume that the reference to the function *random (0,1,2)* will return either the digit 0, the digit 1, or the digit 2, each with probability one-third.

```
let k = length of the bit-string
for j = 1 to k do
    mask[j] = random (0,1,2)
    inverse_mask_1[j] = (mask[j] + 1) MOD 3
    inverse_mask_2[j] = (mask[j] + 2) MOD 3
```

Figure 2. 3-parent uniform crossover and inverse mask construction

Theorem 2: If three children are produced from three parents using the 3-parent uniform crossover mask and its inverses, then all bit-level genetic information is maintained during the crossover portion of the reproduction process.

Proof: Let S_j represent the set resulting from the union of the crossover and two inverse mask values for a given bit-position j. If the cardinality of S_j is 3 for every bit-position j, then no genetic information can be lost because each parent contributes a bit-value to a child. If the crossover mask has a value of 0 for any position j, then one of the inverse masks will have (0 + 1) MOD 3 = 1 in position j and the other inverse mask will have (0 + 2) MOD 3 = 2. If the crossover mask has a value of 1 for any position j, then one of the inverse masks will have (1 + 1) MOD 3 = 2 in position j and the other inverse mask will have (1 + 2) MOD 3 = 0. If the crossover mask has a value of 2 for any position j, then one of the inverse masks will have (2 + 1) MOD 3 = 0 in position j and the other inverse mask will have

(2 + 2) MOD 3 = 1. Regardless of the value in position j of the crossover mask, the cardinality of S_j is 3 and no bit-level genetic information can be lost. Q.E.D.

Here is an example of reproduction using the 3-parent uniform crossover operator.

Parent 0: 011101

Parent 1: 101010

Parent 2: 001100

Mask: 102021

Inverse Mask 1: 210102

Inverse Mask 2: 021210

Child 0: 111100

Child 1: 001000

Child 2: 001111

It is assumed that the two masks are generated using the algorithm shown in Figure 2. As with the 2-parent uniform crossover example above, the children are decidedly different than the parents. Enumeration of the bit-level values for the parents shows that there are nine 1-bits and nine 0-bits. As expected from Theorem 2, enumeration of the bit-level values for the children shows nine 1-bits and nine 0-bits.

The 3-parent uniform crossover operator, along with the mutation operator, is used in the reproduction process as described above.

D. EXPERIMENTATION

1. Problem Set. Functions *F1* through *F5* from the De Jong test suite [8] were used in this research. These functions, along with their corresponding range of x_i values, are given in Table I.

Table I. De Jong Test Suite

F1	$f_1(x_i) = \sum_{i=1}^{3} x_i^2,$	$-5.12 \le x_i \le 5.12$
F2	$f_2(x_i) = 100(x_1^2 - x_2)^2 + (1 - x_1)^2,$	$-2.048 \le x_i \le 2.048$
F3	$f_3(x_i) = \sum_{i=1}^{5} \text{integer}(x_i),$	$-5.12 \le x_i \le 5.12$
F4	$f_4(x_i) = \sum_{i=1}^{30} ix_i^4 + \text{Gauss}(0,1),$	$-1.28 \le x_i \le 1.28$
F5	$f_5(x_i) = 0.002 + \sum_{j=1}^{25} \frac{1}{j + \sum_{i=1}^{2} (x_i - a_{ij})^6}$	$-65.536 \le x_i \le 65.536$

As noted by David Goldberg [3], these functions, which have become standards used to benchmark and compare performances of GAs, include the following characteristics: continuous/discontinuous, convex/nonconvex, unimodal/multimodal, quadratic/nonquadratic, low-dimensionality/high-dimensionality, and deterministic/stochastic. Clearly, not all of the characteristics occur in a single test function.

Because this research was intended to lay a foundation for a new family of GAs, it was thought to be most appropriate to remain "pure" by using De Jong's original encoding scheme (and not the Gray coding used by some GA researchers).

2. GAs with Uniform Crossover. The objective of this study was to compare the newly developed 3-parent uniform crossover operator with the standard 2-parent uniform crossover operator. The GAs employed in this research used both uniform crossover and mutation in the reproduction process. Mutation played a minor role in the final analysis because of the small probability of its occurrence.

Selection for the reproduction process was implemented as a biased roulette wheel. Each individual population member was allocated an amount of the roulette wheel proportional to its objective function value. A uniformly-distributed pseudo-random number between 0 and 1 was generated and compared to the cumulative distribution of values from the weighted roulette wheel. An individual was selected for reproduction when the pseudo-random number fell within that individual's range of values from the cumulative distribution function.

This research used generational replacement as the population replacement strategy. This means that all n population members in generation t were replaced in generation $t+1$. An exception to this would be if an individual was cloned into the next generation as a result of not invoking the crossover operator (the probability of crossover was always less than unity), although being cloned in this manner is not related to the population replacement strategy. The obvious downside to this strategy is that an exceptional individual might be lost early in the search. However, other population replacement strategies allow some individuals to have the god-like characteristic of immortality.

The random number generator is self-contained in the program to ensure replicability of the experiments. The random number generator used is based on L'Ecuyer's Minimum Standard [9], which was shown by Martina Schollmeyer to be both efficient and reliable [10].

As mentioned above, this research was intended to lay a foundation for a new family of GAs. Although there are alternative selection schemes and population replacement strategies which might work better under certain conditions, it is important to note that the GA characteristics used in this research were consistent for both the 2-parent and 3-parent uniform crossover implementations. Therefore, both GAs suffered/benefitted equally from the choice of characteristics.

3. Parameter Settings. Each of the five test functions were used to experiment with GAs using the 2-parent uniform crossover operator and GAs using the 3-parent uniform crossover operator. Experiments were performed using all possible combinations of parameters settings given in Figure 3.

Parameter	Value(s)
Probability of crossover	0.6, 0.7, 0.8, 0.9
Probability of mutation	0.01
Maximum number of generations	50, 100, 150, 200
Population size	60, 120, 180, 240
Number of trials	20

Figure 3. Parameter Settings

A limited number of experiments were also performed with mutation probabilities of 0.0001, 0.001, and 0.05. The mutation probability of 0.01 consistently gave the best results, so it was used for all remaining experiments. The use of a single value for the mutation probability is justifiable because mutation plays such a minor role in the reproduction process.

The reproduction process used in this research generates *m* children from *m* parents. Since population sizes needed to be equal for comparison purposes, it was necessary to have them be multiples of both 2 and 3.

For every combination of the first four parameters listed in Figure 3, 20 trials were performed. All results presented are averages of the 20 trials.

E. RESULTS

The best function value during an execution of a GA (for a given set of parameters) was saved and reported as the best of that trial. Twenty trials were performed for each set of parameters. The average of the twenty "best of trial" values was used to determine if the particular GA was a winner.

Figure 4 shows one of the 80 graphs used to determine the winner. The population size and maximum generation value were held constant and the probability of crossover iterated from 0.6 to 0.9 (inclusive) by 0.1. The best result for all of the crossover probabilities for the 3-parent GA was compared to the best result for all of the crossover probabilities for the 2-parent GA. The winner of this comparison was deemed the winner for that particular set of parameters.

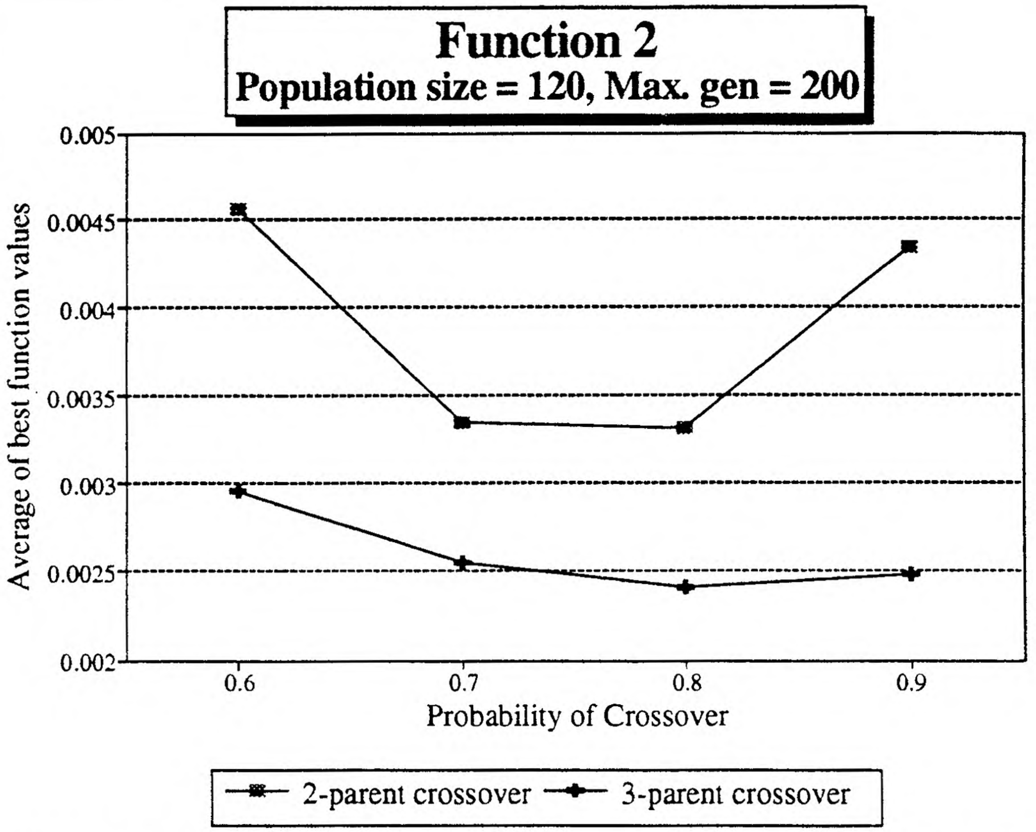

Figure 4. 2-parent vs. 3-parent uniform crossover for a given set of parameters

There were 80 contests (4 population sizes, 4 maximum generation values, and 5 functions). Figure 5 shows the number of wins for the 3-parent crossover GA and the 2-parent crossover GA for a given set of parameters. Overall, the 3-parent GA won 41 of the 80 contests. Functions *F2* and *F5* were clearly dominated by the 3-parent GA, while functions *F1* and *F3* were won by the 2-parent GA (although the margin of victory was not as great with *F1* and *F3* as it was with *F2* and *F5*). While the 2-parent GA did win a majority of the contests using function *F4*, it is clearly not a dominant winner. This margin of victory is too small to make any general statements about which crossover operator is best for *F4*.

Based on this limited sampling of test functions, the GA with 3-parent uniform crossover appears to perform well on functions that are continuous, nonconvex, and of low-dimensionality (*F2* and *F5*). It appears to perform poorly on continuous, convex functions of low-dimensionality (*F1*) and non-continuous functions (*F3*). It performs reasonably well on a convex function of high-dimensionality (*F4*).

Functions *F2* and *F5* are both highly nonlinear and difficult to solve using traditional methods (*F2* is Rosenbrock's function, a classic example from the nonlinear optimization field). These results indicate that the GA with 3-parent uniform crossover will probably perform best on functions that are difficult to solve with traditional methods.

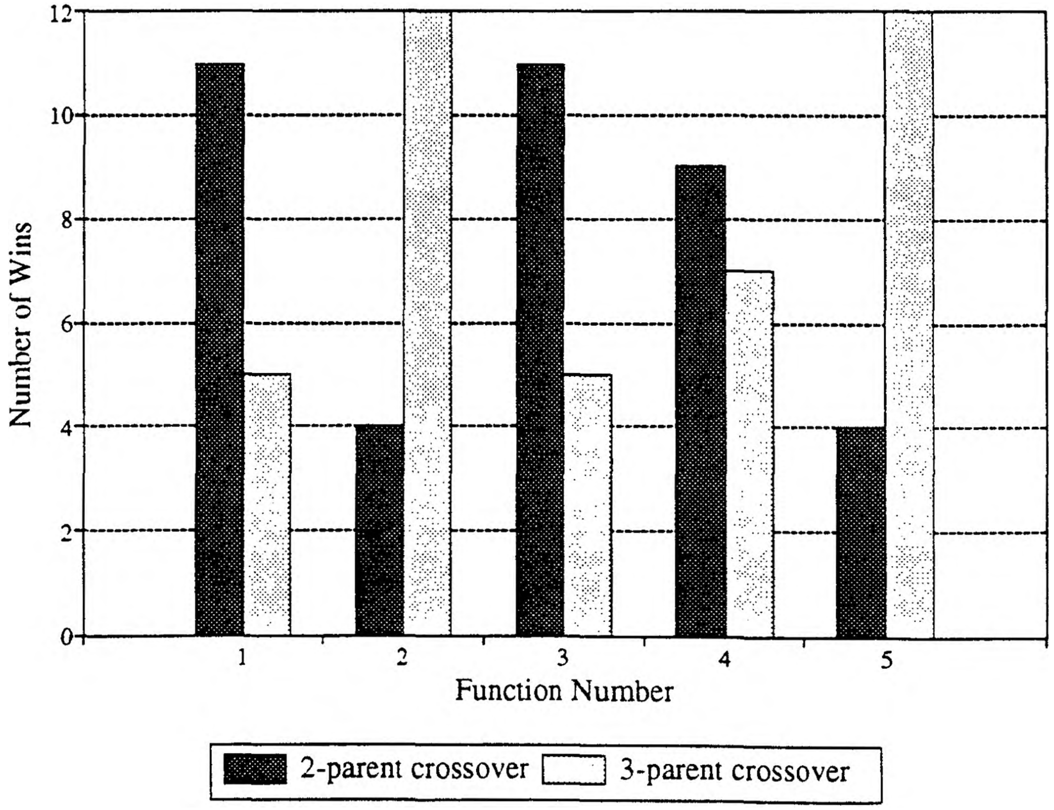

Figure 5. 2-parent vs. 3-parent uniform crossover

Figure 6 shows the crossover probability distribution (as a percentage) for all of the 3-parent winners for a given function. Recall from Figure 5 that the 3-parent GA did not perform well on functions *F1* and *F3*, so the sample size used was relatively small. Consequently, the results shown in Figure 6 for these two functions are of marginal utility.

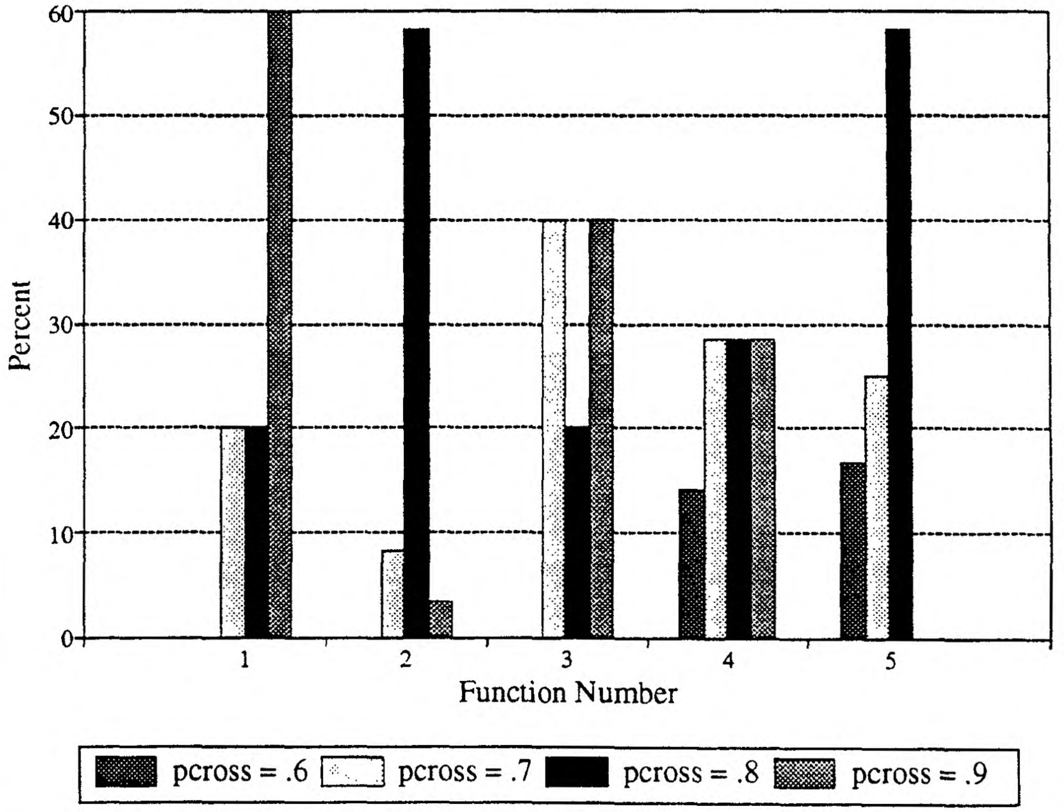

Figure 6. Crossover probability distribution for 3-parent GA winners (by function)

It is useful to make some general observations about parameter settings. Figure 7 shows the crossover probability distribution (as a percentage) for all of the 3-parent GA executions, regardless of the winner. As expected, a relatively large (0.8 - 0.9) crossover probability tends to work best. Uniform crossover has been shown to be disruptive [6], and the more often that it occurs the more the solution space can be explored.

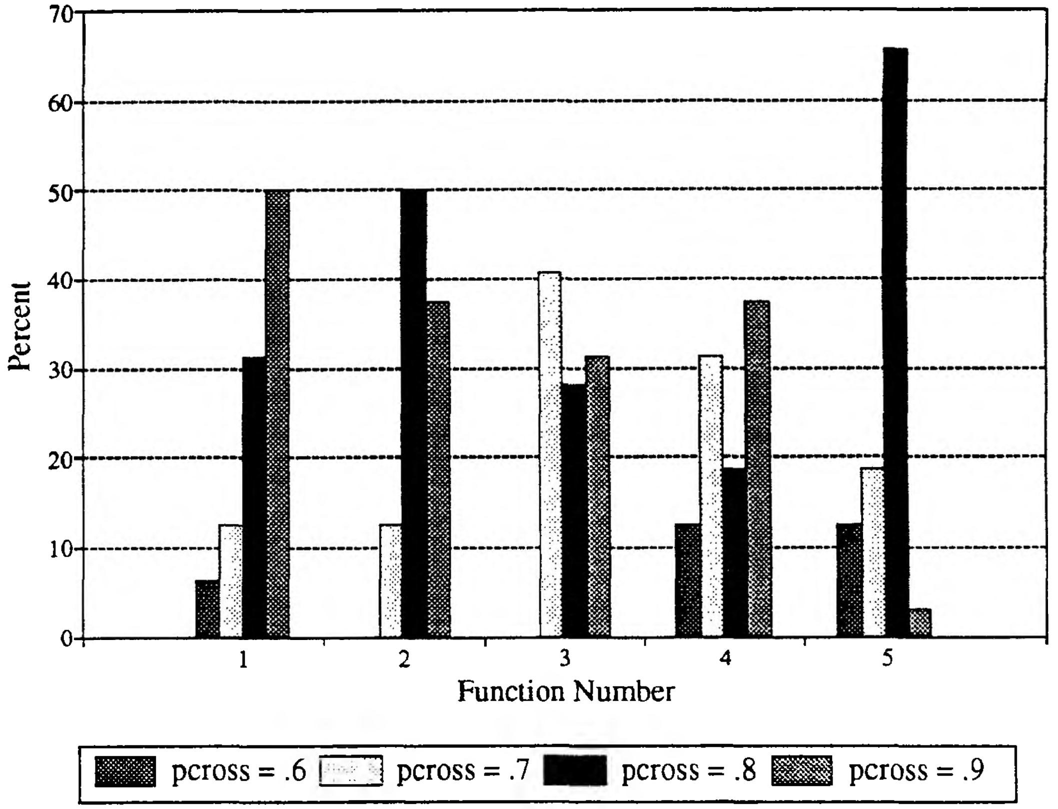

Figure 7. Crossover probability distribution for 3-parent GAs (by function)

Figure 8 strengthens the results from Figure 7 by showing that, regardless of the function being optimized, a large probability of crossover yields better results.

Figure 9 shows the number of winners for both the 2-parent and 3-parent uniform crossover GAs, categorized by the maximum number of generations. Based on these results, it appears that another characteristic of the 3-parent approach is that it performs better with more generations. The category in which the 3-parent approach lost the most to the 2-parent approach was a maximum of 50 generations. This result is not surprising. Intuitively, the 3-parent uniform crossover operator seems more likely than the 2-parent uniform crossover operator to maintain population diversity during the initial part of the search. Stopping the search after only 50 generations would allow a GA that is starting

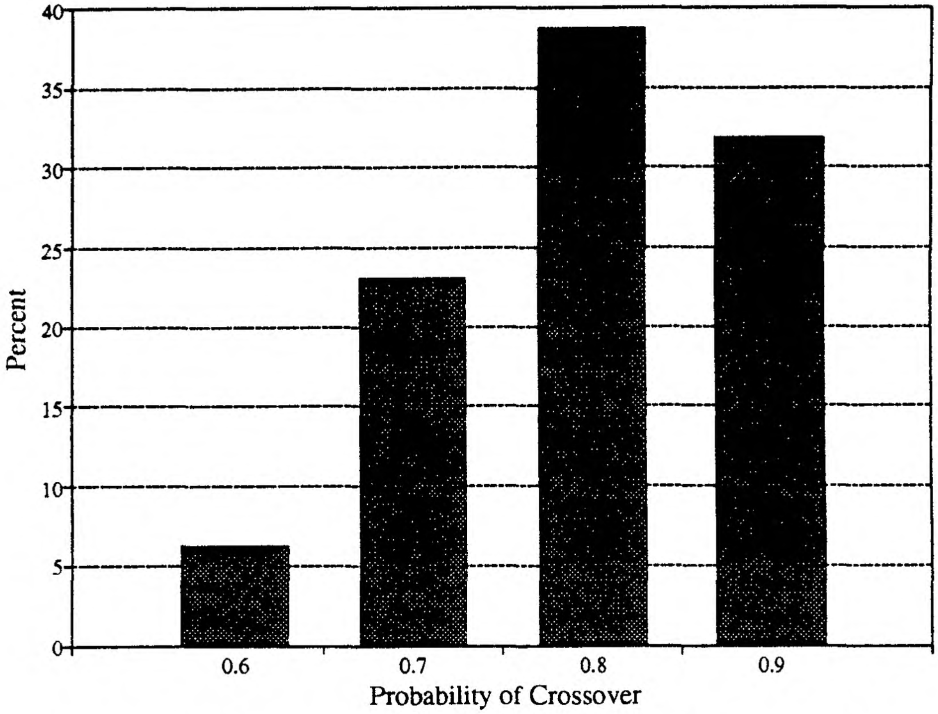

Figure 8. Crossover probability distribution for 3-parent GAs

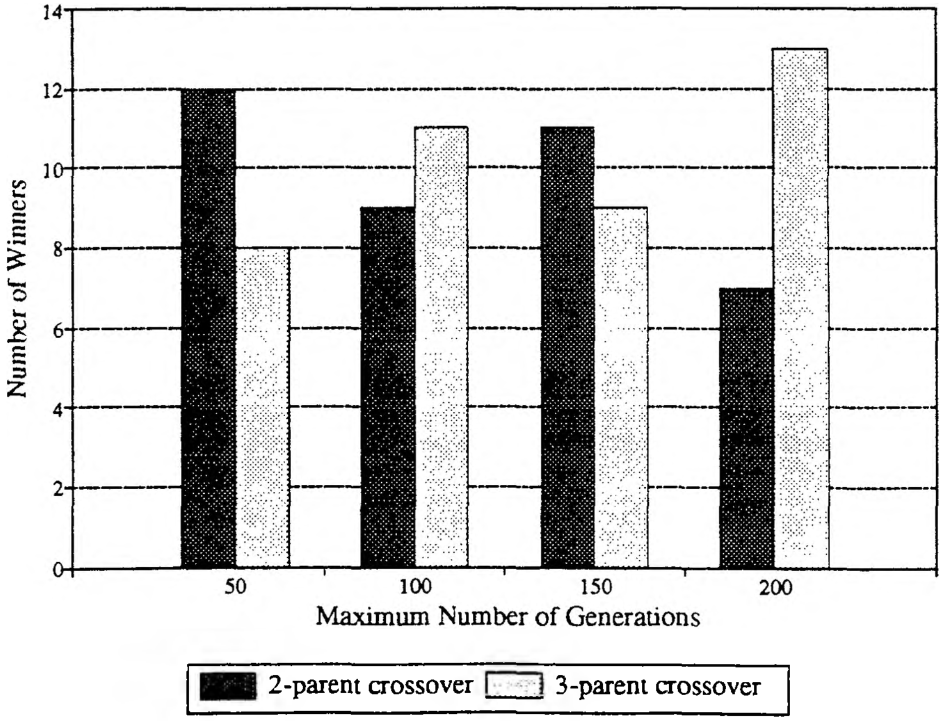

Figure 9. 2-parent vs. 3-parent based on the maximum number of generations

to converge to be deemed the winner, even though it may be converging to a (non-global) local optimum.

As expected, the quality of the solution tends to increase as the number of generations increases. Therefore, the solutions obtained after 200 generations are usually better than those obtained after 50 (or 100 or 150) generations. Consequently, Figure 9 indicates that the GA with 3-parent uniform crossover yields better solutions the majority of the time.

Figure 10 shows the number of winners for both 2-parent and 3-parent uniform crossover GAs, categorized by the population size. Based on these results, it appears that yet another characteristic of the 3-parent approach is that it performs better with a moderate population size. The category in which the 3-parent GA lost to the 2-parent

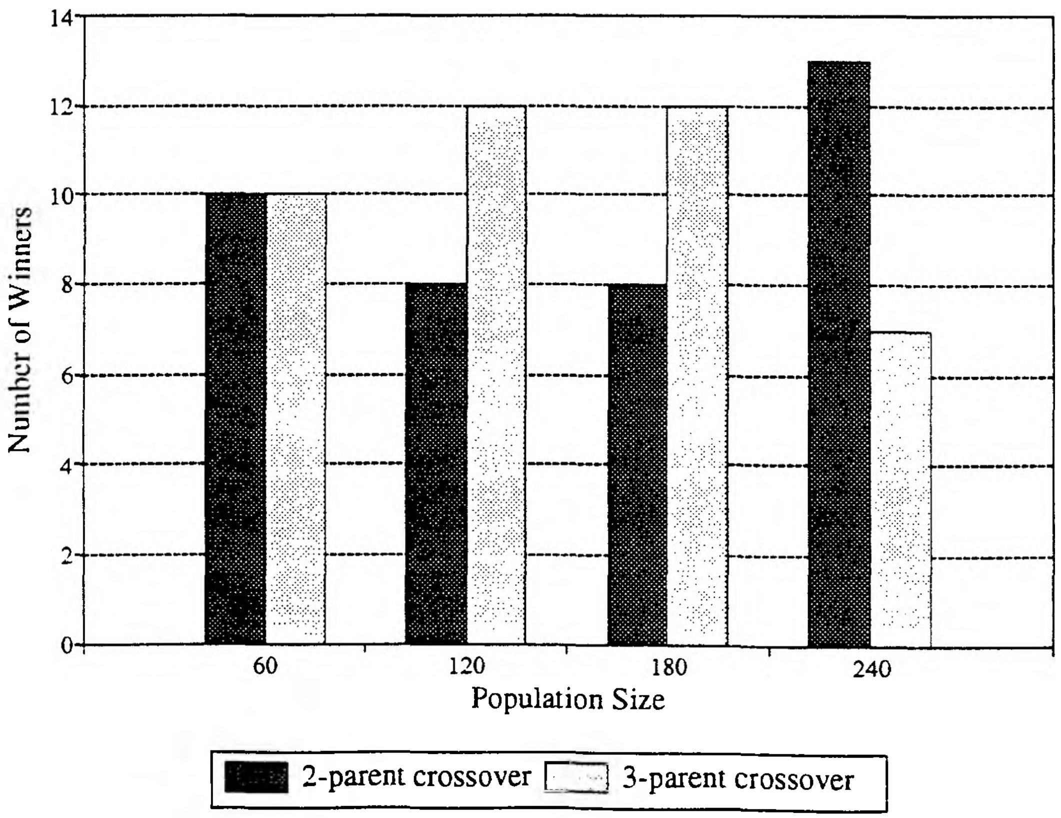

Figure 10. 2-parent vs. 3-parent based on the population size

GA was a population size of 240. It is important to note that many of these losses occurred while the parameter specifying the maximum number of generations was low. Therefore, some of the above comments about a small maximum number of generations apply here as well.

De Jong defined two metrics for GA performance [8]. The on-line performance of a GA is the average of all function evaluations up to and including the current trial. The off-line performance is the average of the best performances up to and including the current trial. Table II shows a sampling of both on-line and off-line performance for each of the five test functions. A crossover strategy was deemed a winner if the majority of function values were less than the corresponding set of function values for the other crossover strategy. Figure 11 gives an example of off-line performance in which the 3-parent approach won.

Table II shows that there is not a clear winner in the on-line and off-line competition between the two crossover strategies. Both the 3-parent and the 2-parent approach yield reasonable (and essentially equal) on-line and off-line performance.

Table II. Sampling of on-line and off-line winners

Function #	Maxgen	Pop. size	Pcross	On-line	Off-line
1	50	60	0.8	3	3
1	100	120	0.7	3	2
1	150	180	0.7	3	~tie
1	200	240	0.8	~tie	2
2	50	60	0.6	3	3
2	100	120	0.7	2	3
2	150	180	0.8	2	3
2	200	240	0.9	3	3
3	50	60	0.9	3	3
3	100	120	0.8	3	3
3	150	180	0.7	2	2
3	200	240	0.6	2	2
4	50	60	0.8	2	2
4	100	120	0.7	2	~tie
4	150	180	0.7	~tie	3
4	200	240	0.8	2	2
5	50	60	0.6	2	~tie
5	100	120	0.7	3	2
5	150	180	0.8	~tie	2
5	200	240	0.9	3	2

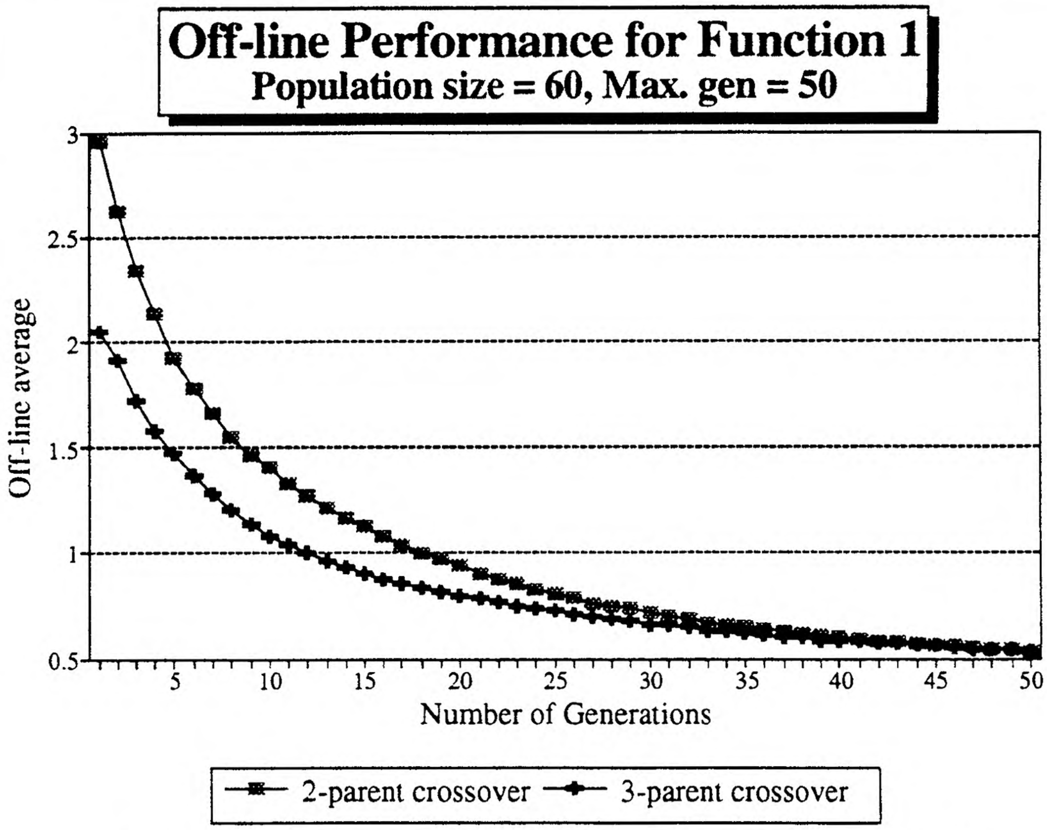

Figure 11. Example of off-line performance for the 3-parent GA

F. CONCLUSION

One of the goals of this research was to lay a foundation for a new family of GAs using a 3-parent uniform crossover operator. Another goal was to obtain better solutions for the De Jong test suite using a GA with 3-parent uniform crossover as compared to a GA with 2-parent uniform crossover. For functions *F2* and *F5*, the 3-parent GA clearly dominates the 2-parent GA. Functions *F1* and *F3* had higher quality solutions when the 2-parent GA was used. Both approaches performed reasonably well on function *F4*.

The data indicate that the 3-parent GA is better suited for continuous functions that are not easily solved with traditional nonlinear optimization techniques. It also appears to be reasonably well-suited for nonlinear functions of high-dimensionality.

As is typical for most GAs, the 3-parent GA solution quality increases as the number of generations increases. It also yields better solutions with a moderate population size. Although the optimal parameter settings are function dependent, the 3-parent GA yields better results with a high crossover probability (≥ 0.8). The data indicate that, overall, the GA with 3-parent uniform crossover is better than the GA with 2-parent uniform crossover.

Another new family of GAs, developed by Vincent Edmondson [11], uses 3-parent traditional crossover operators. These GAs have been shown to be more effective than GAs using 2-parent traditional crossover on all functions in the De Jong test suite except function *F2*. Interestingly, the GA with 3-parent uniform crossover performed well on function *F2*. This suggests that these new families of GAs complement each other and that a 3-parent crossover operator is better than a 2-parent crossover operator. These results provide a firm foundation for the further development of GAs with 3-parent crossover.

G. FUTURE RESEARCH

A future research project using the 3-parent uniform crossover operator might include a selection of functions that are more difficult to optimize than those in the De Jong test suite. Other projects might include the use of alternate selection schemes, alternate population replacement strategies, and parallelization.

Another future research project might involve the development of *n*-parent uniform crossover operators. Clearly, a large value for *n* would just be a random walk through the search space, but it is certainly possible that other *n*-parent uniform crossover GAs, defined in an analogous fashion to the 3-parent GA, could provide better solutions.

REFERENCES

[1] J. H. Holland, *Adaptation in Natural and Artificial Systems*, The University of Michigan Press, Ann Arbor, MI, 1975.

[2] H. Schwefel, *Numerical Optimization of Computer Models*, John Wiley & Sons, New York, NY, 1981.

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

[4] D. H. Ackley, *A Connectionist Machine for Genetic Hillclimbing*, Kluwer Academic Publishers, Boston, MA, 1987.

[5] G. Syswerda, *Uniform Crossover in Genetic Algorithms*, in Proceedings of the Third International Conference on Genetic Algorithms, J. D. Schaffer, ed., Morgan Kaufmann, San Mateo, CA, 1989, 2-9.

[6] L. J. Eshelman, R. A. Caruana, and J. D. Schaffer, *Biases in the Crossover Landscape*, in Proceedings of the Third International Conference on Genetic Algorithms, J. D. Schaffer, ed., Morgan Kaufmann, San Mateo, CA, 1989, 10-19.

[7] W. M. Spears and K. A. De Jong, *On the Virtues of Parameterized Uniform Crossover*, in Proceedings of the Fourth International Conference on Genetic Algorithms, R. K. Belew and L. B. Booker, eds., Morgan Kaufmann, San Mateo, CA, 1991, 230-236.

[8] K. A. De Jong, *An Analysis of the Behavior of a Class of Genetic Adaptive Systems*, Doctoral Thesis, University of Michigan, Ann Arbor, MI, 1975.

[9] P. L'Ecuyer, *Efficient and Portable Combined Random Number Generators*, Communications of the ACM, 31 (1988), 742-749, 774.

[10] M. Schollmeyer, *Noise Generators for Use in Computer Simulation*, M.S. Thesis, University of Missouri-Rolla, Rolla, MO, 1989.

[11] L. V. Edmondson, *Genetic Algorithms with 3-parent Crossover*, Ph.D. Dissertation, University of Missouri-Rolla, Rolla, MO, 1993.

II. GENETIC ALGORITHMS WITH 3-PARENT TRADITIONAL CROSSOVER

A. ABSTRACT

New genetic algorithms which use 3-parent traditional crossover operators are presented. The goal of the research was to obtain better results for the De Jong test functions using the 3-parent traditional crossover operators in comparison to the 2-parent traditional crossover operator. Each of the 3-parent traditional crossover operators is shown to be superior to the 2-parent traditional crossover operator. The genetic algorithm using 3-parent traditional crossover and a strategy of choosing the best 3 out of 6 children resulting from 3-parent reproduction is shown to be superior to all other genetic algorithms considered in this research.

B. INTRODUCTION

Genetic algorithms (GAs) are randomized search procedures which apply the Darwinian notion of survival of the fittest to a population of individuals. These algorithms were developed independently by John Holland at the University of Michigan [1] and by Ingo Rechenberg and Hans-Paul Schwefel in Germany [2]. The fields to which GAs have been applied are numerous. They range from engineering and computer science to the social sciences [3]. It is anticipated that, because of their robust nature, GAs will continue to be applied to a wide variety of areas.

The traditional genetic algorithm (GA), as developed by Holland, begins with a population of randomly-generated binary string creatures. The fitness of each individual in the population is evaluated using an objective function and then these objective function values are used to determine which individuals will participate in the reproduction process. Selection for the reproduction process can be easily understood as a biased roulette wheel. Each individual is allocated an amount of the roulette wheel which is proportional to its objective function value. The actual reproduction process involves the two operators of crossover and mutation. The crossover operator exchanges bits (genetic information) between two parents. The mutation operator (which is invoked with only a small probability) is used to change a 0 to 1 or a 1 to 0. This perturbation is used to ensure that population diversity is maintained. This reproduction process is used to create a new generation of population members. The fitness of each individual in the new generation is then evaluated and the aforementioned process is repeated for either a preset number of generations or a preset amount of computer time.

C. TRADITIONAL CROSSOVER OPERATORS

The traditional crossover operator was originally developed by Holland [1]. Although other types of crossover operators, such as uniform and order-based crossover, have been developed, traditional crossover remains the predominant choice. All four of the international conferences on GAs include papers dealing with traditional crossover [4,5,6,7].

1. 2-parent Traditional Crossover. The 2-parent traditional crossover operator uses a crossover mask. This crossover mask is a string of bits in which the parity of each bit determines which parent will contribute the genetic information to the child. Each crossover mask has an inverse mask in which the parity of each bit in the crossover mask is reversed. For example, if a crossover mask is 11100, then its inverse mask is 00011. This is an example of 2-parent, 1-point crossover. A crossover point (position 3 in the previous example) determines the position from which bit-level genetic information will start to be contributed from the other parent.

It has been shown [8,9] that 2-parent, 2-point crossover is superior to 2-parent, 1-point crossover. Therefore, all subsequent references to 2-parent crossover will actually be for 2-parent, 2-point crossover. An algorithm for constructing a 2-parent crossover mask and its inverse is given in Figure 12. Assume that the reference to the function *random (k-1)* will sample from the uniform distribution and will return an integer in the range from *1* to *k-1* (inclusive).

```
let k = length of the bit-string
t1 = random (k-1)
t2 = random (k-1)
if t1 > t2 then
        exchange t1 and t2
for j = 1 to t1 do
        mask[j] = 0
        inverse_mask[j] = 1
for j = (t1+1) to t2 do
        mask[j] = 1
        inverse_mask[j] = 0
for j = (t2+1) to k do
        mask[j] = 0
        inverse_mask[j] = 1
```

Figure 12. 2-parent traditional crossover and inverse mask construction

Here is an example of reproduction using the 2-parent traditional crossover operator. Assume, without loss of generality, that the crossover points are in positions 2 and 4.

Parent 0: 011101

Parent 1: 101010

Mask: 001100

Inverse Mask: 110011

Child 0: 011001

Child 1: 101110

It is assumed that the two masks are generated using the algorithm shown in Figure 12. As is typical for traditional crossover, the children are very similar to the parents. The 2-parent traditional crossover operator, along with the mutation operator, is used in the reproduction process as described above.

2. 3-parent Traditional Crossover. The 3-parent traditional crossover operators are new reproduction operators that are generalizations of the 2-parent traditional crossover operator. They use crossover masks that allow 3 parents to pass along genetic information to a child. Although the idea of using 3 parents for reproduction is not based in nature (and, hence, the Zen *koan* of letting nature be the guiding principle of GA design is violated [10]), the 3-parent approach is an interesting abstraction of the standard 2-parent reproduction process.

In order for all 3 parents to contribute this genetic information, a minimum of 2 crossover points is required. Let **0**, **1**, and **2** represent strings of 0's, 1's, and 2's, respectively, to be used in crossover masks. There are 3! possible masks: **012**, **021**, **102**, **120**, **201**, and **210**. Under the assumption that *n* parents should generate *n* children, a strategy needs to be developed for reproducing 3 children that will survive into the next generation.

a. 3-parent traditional crossover with random 3 of 6 children. One strategy for reproducing 3 children from 3 parents is to define crossover so that it randomly chooses 3 of 6 children. The idea of an inverse mask is not well-defined when using 3-parent traditional crossover. Therefore, the algorithm given in Figure 13 creates 3 crossover masks without reference to an inverse. The variable *v* represents a set which can hold integer values in the range from 1 to 6 (inclusive). Assume that the reference to the function *random (6)* will sample from the uniform distribution and will return an integer in the range from *1* to *6* (inclusive). Assume also that the crossover points are randomly generated values and that the mask notation is consistent with the notation defined above.

```
v = []
for j = 1 to 3 do
        k = random (6)
        while k in v do
                k = random (6)
        end while
        v = v + [k]
        case k of
                1 : mask[j] = 012
                2 : mask[j] = 021
                3 : mask[j] = 102
                4 : mask[j] = 120
                5 : mask[j] = 201
                6 : mask[j] = 210
        end case
end for
```

Figure 13. 3-parent traditional crossover mask construction for random 3 of 6 children

b. 3-parent traditional crossover with best 3 of 6 children. Continuing with the assumption that 3 children should come from 3 parents, another way to define crossover using 3 parents and 2 crossover points is to generate all 6 children, but only allow the best 3 to survive into the next generation. Although this would be an abhorrence if applied to humans, in the artificial world of GAs it is merely a small-scale survival-of-the-fittest algorithm. Mathematically, it is a local optimization procedure which is applied after each set of parents reproduces. Figure 14 gives the algorithm for determining which 3 children will survive. Without loss of generality, assume that the objective function is to be minimized.

```
create all 6 children with masks 012, 021, 102, 120, 201, and 210
evaluate each of the six children using the objective function
sort the function values into ascending order
keep the children corresponding to the first 3 elements of the sorted array
```

Figure 14. 3-parent traditional crossover for best 3 of 6 children

Here is an example of reproduction using 3-parent traditional crossover operator masks. The strategy for selecting the survivors will have no impact on the method of generating the children.

Parent 0: 011101
Parent 1: 101010
Parent 2: 001100
Mask 0: 001222
Mask 1: 220111
Mask 2: 110222
Child 0: 011100

Child 1: 001010

Child 2: 101100

D. EXPERIMENTATION

1. Problem Set. Functions *F1* through *F5* from the De Jong test suite [11] were used in this research. These functions, along with their corresponding range of x_i values, are given in Table III.

Table III. De Jong Test Suite

F1	$f_1(x_i) = \sum_{i=1}^{3} x_i^2,$	$-5.12 \leq x_i \leq 5.12$
F2	$f_2(x_i) = 100(x_1^2 - x_2)^2 + (1 - x_1)^2,$	$-2.048 \leq x_i \leq 2.048$
F3	$f_3(x_i) = \sum_{i=1}^{5} \text{integer}(x_i),$	$-5.12 \leq x_i \leq 5.12$
F4	$f_4(x_i) = \sum_{i=1}^{30} i x_i^4 + \text{Gauss}(0,1),$	$-1.28 \leq x_i \leq 1.28$
F5	$f_5(x_i) = 0.002 + \sum_{j=1}^{25} \frac{1}{j + \sum_{i=1}^{2} (x_i - a_{ij})^6}$	$-65.536 \leq x_i \leq 65.536$

As noted by David Goldberg [3], these functions, which have become standards used to benchmark and compare performances of GAs, include the following characteristics: continuous/discontinuous, convex/nonconvex, unimodal/multimodal, quadratic/nonquadratic, low-dimensionality/high-dimensionality, and deterministic/stochastic. Clearly, not all of the characteristics occur in a single test function.

Because this research was intended to lay a foundation for a new family of GAs, it was thought to be most appropriate to remain "pure" by using De Jong's original encoding scheme (and not the Gray coding used by some GA researchers).

2. GAs with Traditional Crossover. The objective of this study was to compare the newly developed 3-parent traditional crossover operators with the standard 2-parent traditional crossover operator. The GAs employed in this research used both traditional crossover and mutation in the reproduction process. Mutation played a minor role in the final analysis because of the small probability of its occurrence.

Selection for the reproduction process was implemented as a biased roulette wheel. Each individual population member was allocated an amount of the roulette wheel proportional to its objective function value. A uniformly-distributed pseudo-random number between 0 and 1 was generated and compared to the cumulative distribution of values from the weighted roulette wheel. An individual was selected for reproduction when the pseudo-random number fell within that individual's range of values from the cumulative distribution function.

This research used generational replacement as the population replacement strategy. This means that all n population members in generation t were replaced in generation $t+1$. An exception to this would be if an individual was cloned into the next generation as a result of not invoking the crossover operator (the probability of crossover was always less than unity), although being cloned in this manner is not related to the population replacement strategy. The obvious downside to this strategy is that an exceptional individual might be lost early in the search. However, other population replacement strategies allow some individuals to have the god-like characteristic of immortality.

The random number generator is self-contained in the program to ensure replicability of the experiments. The random number generator used is based on L'Ecuyer's Minimum Standard [12], which was shown by Martina Schollmeyer to be both efficient and reliable [13].

As mentioned above, this research was intended to lay a foundation for a new family of GAs. Although there are alternative selection schemes and population replacement strategies which might work better under certain conditions, it is important to note that the GA characteristics used in this research were consistent for both the 2-parent and 3-parent traditional crossover implementations. Therefore, all GAs suffered/benefitted equally from the choice of characteristics.

3. Parameter Settings. Each of the five test functions were used to experiment with GAs using the 2-parent traditional crossover operator and GAs using the 3-parent traditional crossover operators. Experiments were performed using all possible combinations of parameters settings given in Figure 15.

In addition to these experiments, the GA with 3-parent traditional crossover using the best 3 out of 6 children was executed with a maximum of 25 generations. The purpose of this was to allow for a fair comparison based on the actual number of function evaluations. For a population of size n, each of the other two approaches evaluated the objective function n times, while the "best 3 of 6" approach evaluated the objective function $2n$ times.

Parameter	Values
Probability of crossover	0.6, 0.7, 0.8, 0.9
Probability of mutation	0.01
Maximum number of generations	50, 100, 150, 200
Population size	60, 120, 180, 240
Number of trials	20

Figure 15. Parameter Settings

A limited number of experiments were performed with mutation probabilities of 0.001 and 0.01. The mutation probability of 0.01 consistently gave the best results, so it was used for all remaining experiments. The use of a single value for the mutation probability is justifiable because mutation plays such a minor role in the reproduction process.

The reproduction process used in this research generates *m* children from *m* parents. Since population sizes needed to be equal for comparison purposes, it was necessary to have them be multiples of both 2 and 3.

For every combination of the first four parameters listed in Figure 15, 20 trials were performed. All results presented are averages of the 20 trials.

E. RESULTS

The best function value during an execution of a GA (for a given set of parameters) was saved and reported as the best of that trial. Twenty trials were performed for each set of parameters. The average of the twenty "best of trial" values was used to determine if the particular GA was a winner. The GA using 2-parent traditional crossover is compared separately with the two 3-parent approaches.

1. 2-parent versus 3-parent using random 3 of 6 children. Figure 16 shows one of the 80 graphs used to determine the winner. The population size and maximum generation value were held constant and the probability of crossover iterated from 0.6 to 0.9 (inclusive) by 0.1. The best result for all of the crossover probabilities for the 3-parent GA was compared to the best result for all of the crossover probabilities for the 2-parent GA. The winner of this comparison was deemed the winner for that particular set of parameters.

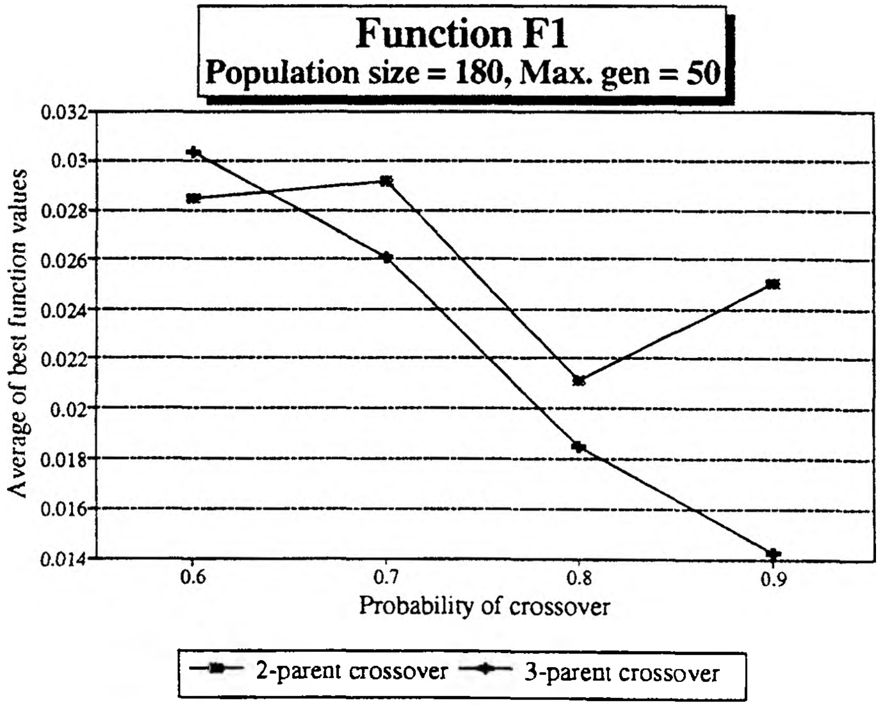

Figure 16. 2-parent vs. 3-parent (random 3 of 6 children) traditional crossover for a given set of parameters

There were 80 contests (4 population sizes, 4 maximum generation values, and 5 functions). Figure 17 shows the number of wins for the 3-parent traditional crossover GA using a random 3 out of 6 strategy and the 2-parent traditional crossover GA for a given set of parameters. Overall, the 3-parent GA won 57 of the 80 contests. The 3-parent GA won a majority of the contests for functions *F1*, *F2*, *F3*, and *F5*, and tied with the 2-parent GA for function *F4*. Each of the functions *F1*, *F2*, *F3*, and *F5* was clearly dominated by the 3-parent GA. It is not possible to make any general statements about which crossover operator is best for function *F4*.

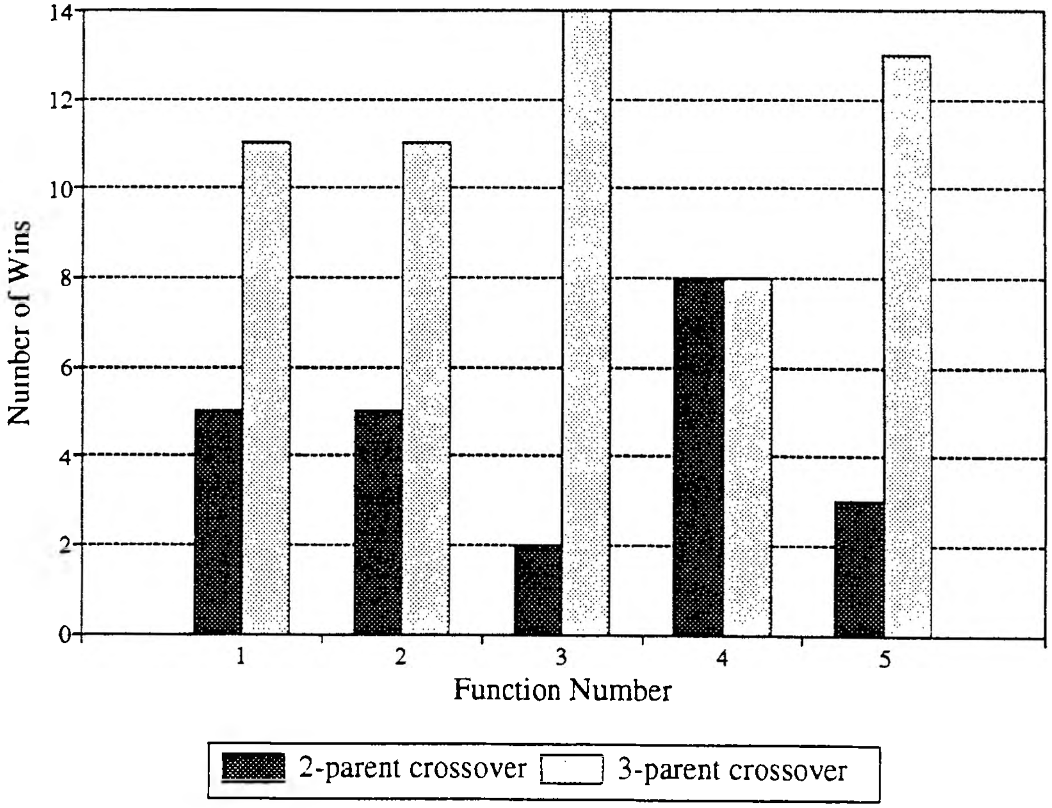

Figure 17. 2-parent vs. 3-parent (random 3 of 6 children) traditional crossover

Based on this limited sampling of test functions, the GA with 3-parent traditional crossover appears to perform exceptionally well on functions that are continuous and of low-dimensionality, regardless of convexity (*F1*, *F2*, and *F5*). It also appears to perform exceptionally well on non-continuous functions (*F3*). Results for continuous, convex functions of high-dimensionality are mixed (*F4*).

Figure 18 shows the crossover probability distribution (as a percentage) for all of the 3-parent winners for a given function. Recall from Figure 17 that the 3-parent GA did not win a majority of the contests using function *F4*, so the sample size used was relatively small. Consequently, the results shown in Figure 18 for this function are of marginal utility.

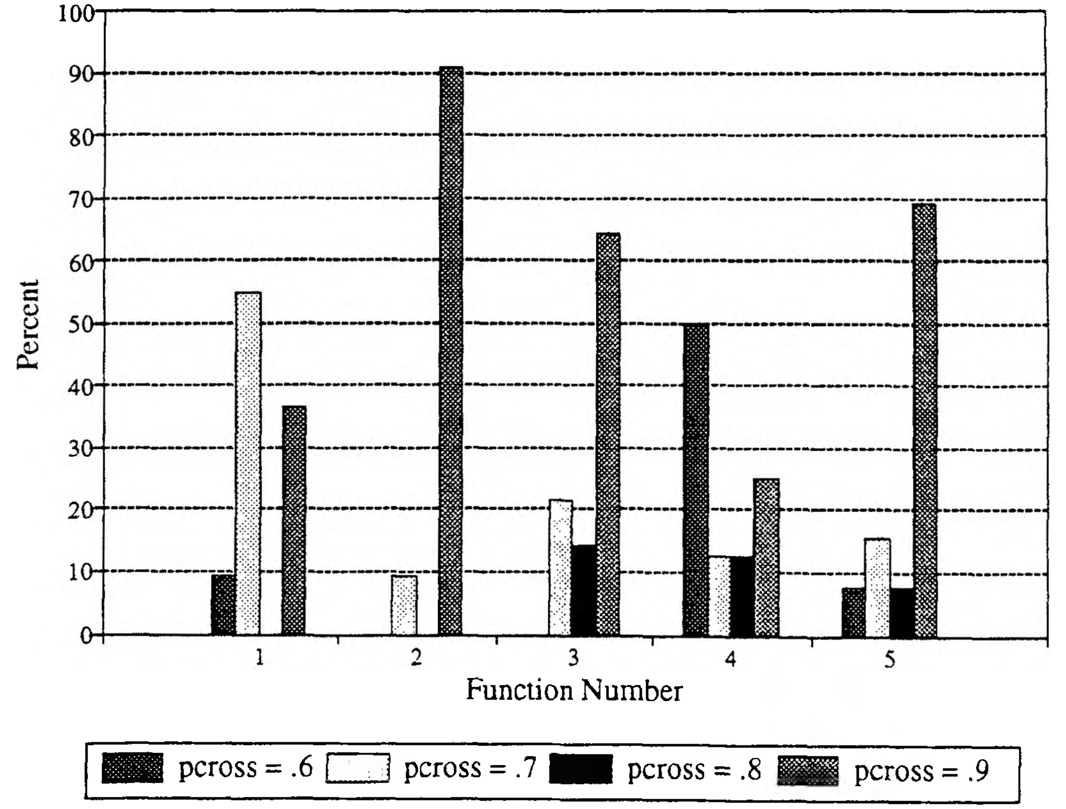

Figure 18. Crossover probability distribution for 3-parent GA winners (by function)

It is useful to make some general observations about parameter settings. Figure 19 shows the crossover probability distribution (as a percentage) for all of the 3-parent GA executions, regardless of the winner. Interestingly, these distributions appear to be bimodal. The crossover probability should either be high (0.9), indicating that crossover occurs frequently and the solution space is more thoroughly explored, or be relatively low (0.6 - 0.7), indicating that curren' solutions are better than solutions that could be reached via crossover.

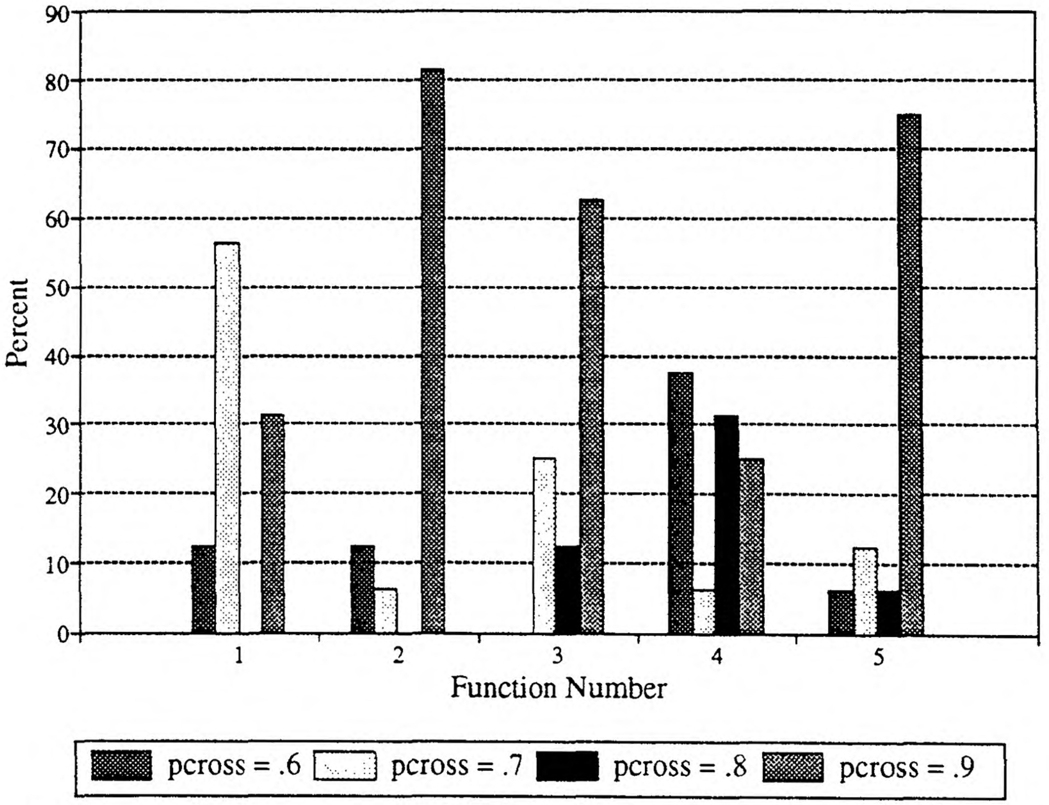

Figure 19. Crossover probability distribution for 3-parent GAs (by function)

Figure 20 strengthens the results from Figure 19 by showing that, overall, the crossover probability distribution is bimodal. The data indicate that, although the optimal settings are function dependent, it is reasonable to begin with a high crossover probability.

Figure 21 shows the number of winners for both the 2-parent and 3-parent traditional crossover GAs, categorized by the maximum number of generations. Based

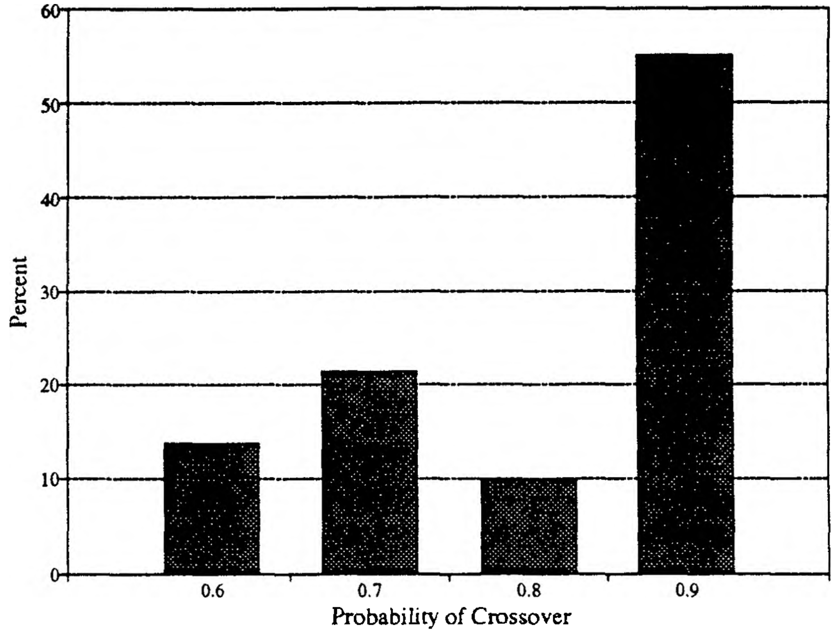

Figure 20. Crossover probability distribution for 3-parent (random 3 of 6 children) GAs

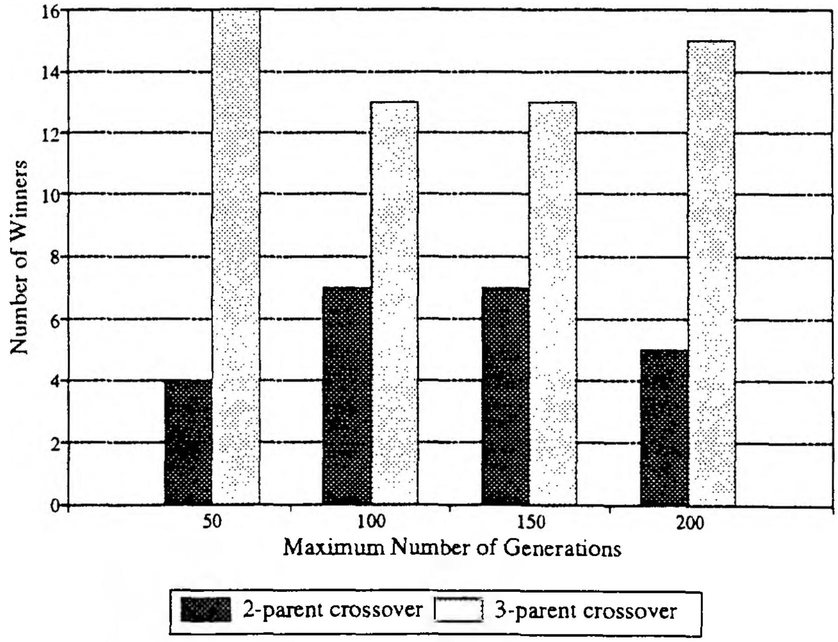

Figure 21. 2-parent vs. 3-parent (random 3 of 6 children) based on the maximum number of generations

on these results, it appears that the 3-parent approach is relatively consistent (and dominant) across all parameter settings for the maximum number of generations.

As expected, the quality of the solution tends to increase as the number of generations increases. Therefore, the solutions obtained after 200 generations are usually better than those obtained after 50 (or 100 or 150) generations. Consequently, Figure 21 indicates that the GA with 3-parent traditional crossover yields better solutions the majority of the time.

Figure 22 shows the number of winners for both 2-parent and 3-parent traditional crossover GAs, categorized by the population size. Based on these results, it appears that another characteristic of the 3-parent approach is that it performs better with a larger population size. The only category in which the 2-parent approach did as well as the 3-parent approach was a population size of 60. Generally, a larger population size results in a higher level of diversity in the population. This higher level of diversity, combined with the more disruptive 3-parent crossover operator, allows more of the solution space to be explored.

2. 2-parent versus 3-parent using best 3 of 6 children. The results from the 2-parent traditional crossover GA were also compared to the results from the 3-parent traditional crossover GA using a strategy of keeping the best 3 out of 6 children. This 3-parent approach gave phenomenal results with a maximum of just 25 generations. Therefore, all comparisons made with this 3-parent approach had a maximum of 25 generations. This means that, for some of the contests, the 2-parent approach was allowed to have as many as 4 times the number of objective function evaluations as the 3-parent approach. Figure 23 shows the number of wins for the 3-parent traditional crossover GA using a best 3 out of 6 strategy and the 2-parent traditional crossover GA for a given set of parameters.

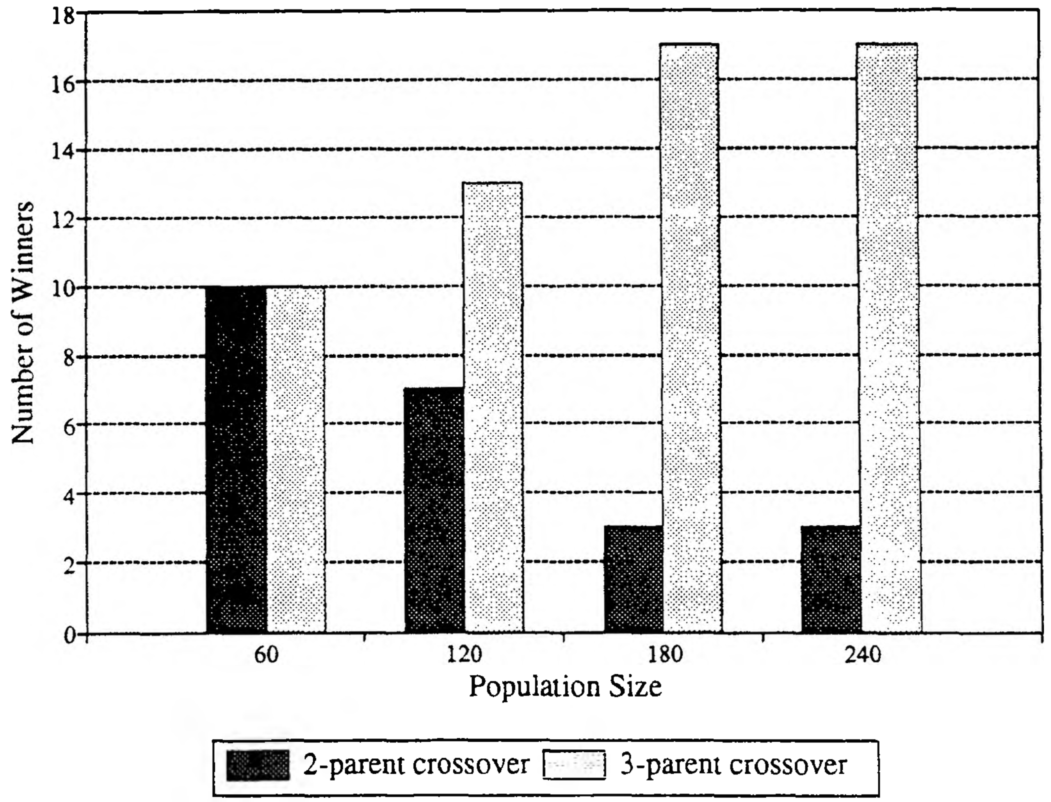

Figure 22. 2-parent vs. 3-parent (random 3 of 6 children) based on population size

Overall, the 3-parent GA won 68 of the 80 contests. The 3-parent GA won a majority of the contests for functions *F1*, *F3*, *F4*, and *F5*, while the majority of the contests for function *F2* were won by the 2-parent GA. With the exception of function *F2*, the 3-parent approach clearly dominated the 2-parent approach, winning a minimum of 15 of the 16 contests for a given function.

Based on this limited sampling of test functions, the GA with 3-parent traditional crossover appears to perform exceptionally well on functions that are continuous and

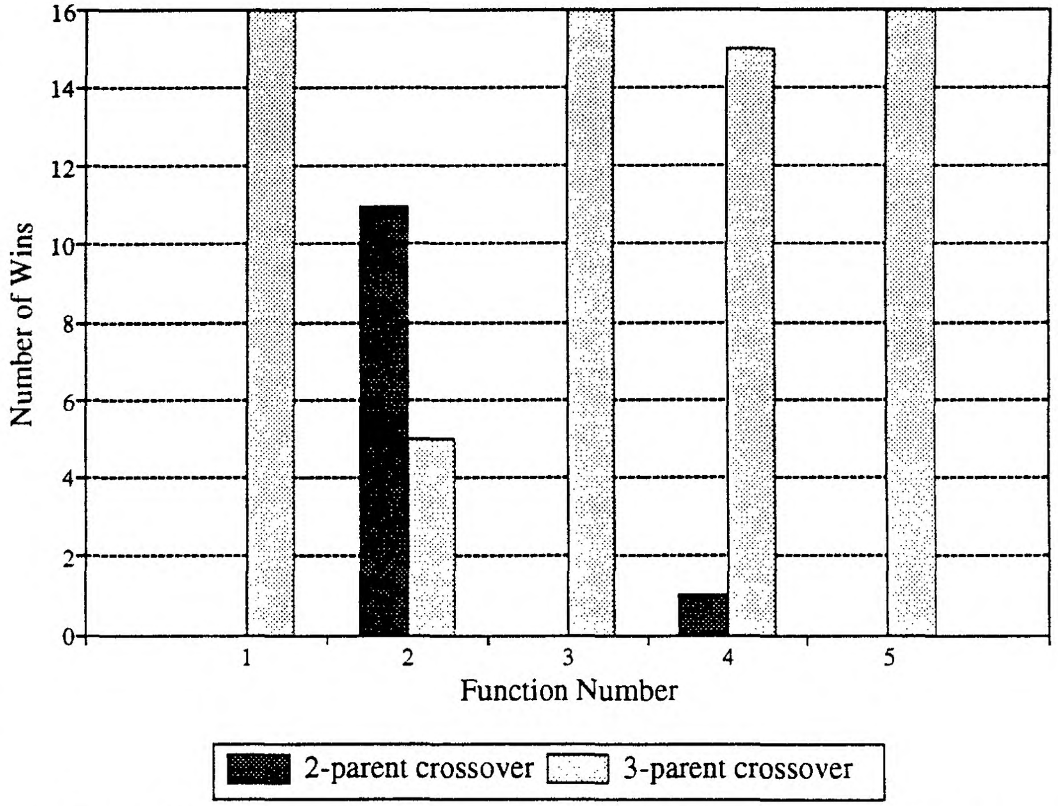

Figure 23. 2-parent traditional crossover vs. 3-parent traditional crossover using best 3 of 6 children

convex, regardless of dimensionality (*F1* and *F4*) and on non-continuous functions (*F3*). Results for continuous, convex functions of low-dimensionality are mixed (*F2* results are poor and *F5* results are good). The poor results on *F2* indicate that the 3-parent approach can be misled by a function which is nonconvex with many local optima. Function *F2* is Rosenbrock's function, a classic example from the nonlinear optimization field. The local optimization which is performed after each set of parents reproduces probably causes this 3-parent approach to become more firmly entrenched in a local optimum, thereby reducing its ability to explore the solution space.

Figure 24 shows the crossover probability distribution (as a percentage) for all of the 3-parent GA executions, regardless of the winner. A high crossover probability (0.9) is clearly the best choice. This indicates that the 3-parent approach performs best when the crossover operator is invoked often, thereby allowing more of the solution space to be searched. It should be noted, however, that this particular 3-parent approach is highly insensitive to the choice of crossover probability. This insensitivity serves to strengthen the robustness of the GA.

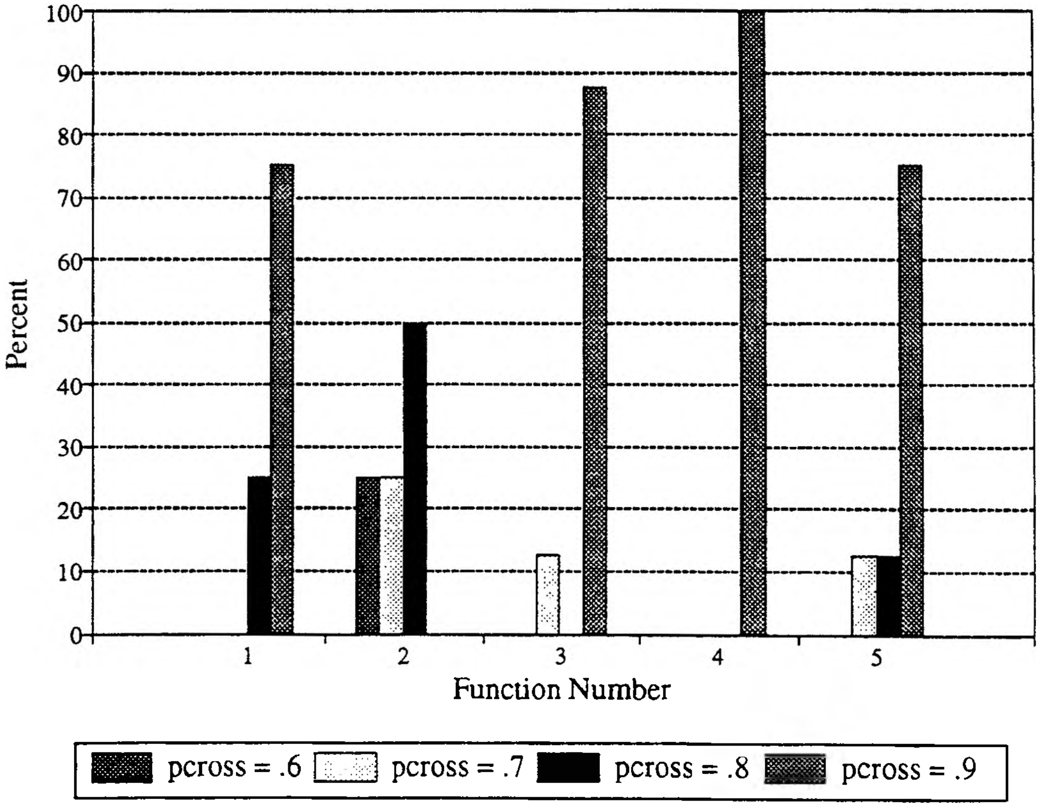

Figure 24. Crossover probability distribution for 3-parent GAs (by function)

As expected, the results for this 3-parent approach were increasingly better as the number of generations increased. It is interesting to note that, if this 3-parent approach is going to work well, it does so after only a small number of generations (25). This makes the algorithm relatively efficient and provides a good basis for determining when it will probably not be fruitful to continue its use.

3. Combined results for all traditional crossover operators.

Figure 25 shows the number of wins for the 3-parent traditional crossover GAs and the 2-parent traditional crossover GA for a given set of parameters. As described above, the 3-parent approach using the best 3 out of 6 strategy for selecting children had a maximum generation count of 25 for all executions. Overall, the 3-parent approaches combined for a total of 75 wins out of the 80 contests. Function *F2* is still the most challenging for the 3-parent approach. These results show the marked superiority of the 3-parent traditional crossover GA.

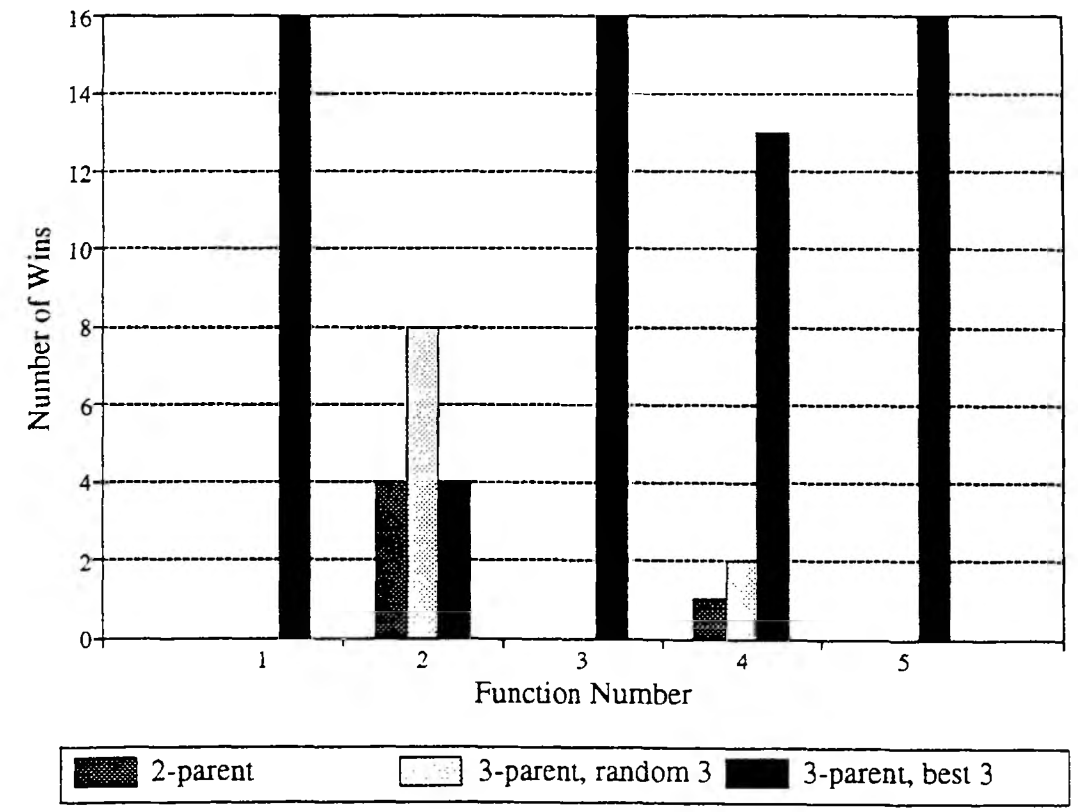

Figure 25. 2-parent traditional crossover vs. both 3-parent traditional crossover operators

4. On-line and off-line performance. De Jong defined two metrics for GA performance [11]. The on-line performance of a GA is the average of all function evaluations up to and including the current trial. The off-line performance is the average of the best performances up to and including the current trial. A sampling of both on-line and off-line performance for each of the five test functions indicates that the GA with 3-parent traditional crossover using the best 3 out of 6 strategy for selecting children is dominant. Interestingly, this approach even had better on-line and off-line performance for function *F2*. This indicates that the population converged quickly to a (non-global) local minimum and was unable to find a better function value after that convergence. Figure 26 gives an example of off-line performance in which the 3-parent approach using the best 3 out of 6 strategy for selecting children won.

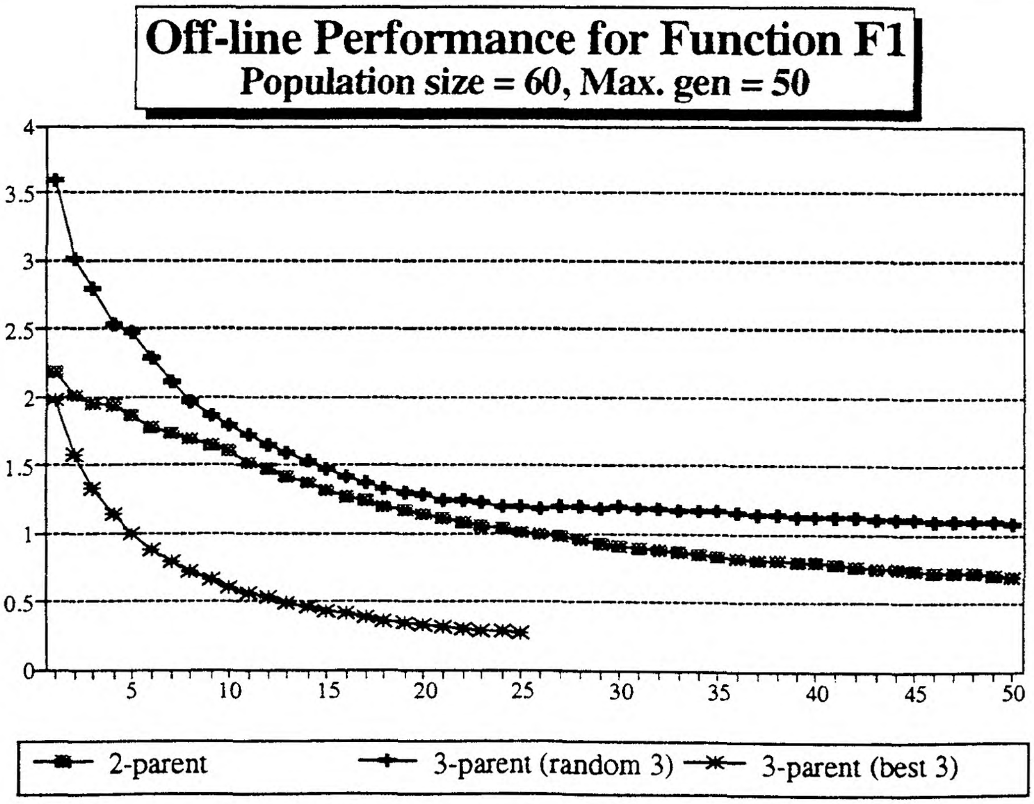

Figure 26. Example of off-line performance

F. CONCLUSION

One of the goals of this research was to lay a foundation for a new family of GAs using 3-parent traditional crossover operators. Another goal was to obtain better solutions for the De Jong test suite using a GA with 3-parent traditional crossover as compared to a GA with 2-parent traditional crossover. The 3-parent GA clearly dominates the 2-parent GA for all functions considered. The 3-parent GA using the best 3 out of 6 strategy of selecting children is better than the 3-parent GA using the random 3 out of 6 strategy.

The data indicate that the 3-parent GA is well suited for both continuous and non-continuous functions of both low-dimensionality and high-dimensionality. Some nonconvex functions can lead the 3-parent GA into a local optimum from which it has difficulty escaping.

The 3-parent GA solution quality increases as the number of generations increases (this is typical for most GAs). A population size larger than 60 also tends to increase the 3-parent GA solution quality. The GA using 3-parent traditional crossover and the best 3 out of 6 strategy for selecting children performs best with a high probability (0.9) of crossover. The GA using 3-parent traditional crossover and the random 3 out of 6 strategy for selecting children is more sensitive to the crossover probability. In spite of this sensitivity, a high probability (0.9) of crossover appears to be a reasonable choice.

The data indicate that, overall, the GA with 3-parent traditional crossover and the best 3 out of 6 strategy for selecting children is markedly superior than GAs using either 2-parent traditional crossover or 3-parent traditional crossover and the random 3 out of 6 strategy for selecting children. This latter 3-parent approach is better than the 2-parent approach.

Another new family of GAs, developed by Vincent Edmondson [14], uses a 3-parent uniform crossover operator. These GAs have been shown to be effective on function *F2* (the single test function on which the GA using 3-parent traditional crossover and the best 3 out of 6 strategy for selecting children performed poorly). This suggests that these new families of GAs complement each other and that a 3-parent crossover operator is better than a 2-parent crossover operator. These results provide a firm foundation for the further development of GAs with 3-parent crossover.

G. FUTURE RESEARCH

A future research project using the 3-parent traditional crossover operators might include a selection of functions that are more difficult to optimize than those in the De Jong test suite. Other projects might include the use of alternate selection schemes, alternate population replacement strategies, and parallelization.

Another future research project might involve the development of n-parent traditional crossover operators. Clearly, a large value for n would just be a random walk through the search space, but it is certainly possible that other n-parent traditional crossover GAs, defined in an analogous fashion to the 3-parent GA, could provide better solutions.

REFERENCES

[1] J. H. Holland, *Adaptation in Natural and Artificial Systems*, The University of Michigan Press, Ann Arbor, MI, 1975.

[2] H. Schwefel, *Numerical Optimization of Computer Models*, John Wiley & Sons, New York, NY, 1981.

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

[4] J. J. Grefenstette, ed., *Proceedings of the First International Conference on Genetic Algorithms and Their Applications*, Lawrence Erlbaum Associates, Hillsdale, NJ, 1985.

[5] J. J. Grefenstette, ed., *Genetic Algorithms and Their Applications: Proceedings of the Second International Conference on Genetic Algorithms*, Lawrence Erlbaum Associates, Hillsdale, NJ, 1987.

[6] J. D. Schaffer, ed., *Proceedings of the Third International Conference on Genetic Algorithms*, Morgan Kaufmann, San Mateo, CA, 1989.

[7] R. K. Belew and L. B. Booker, eds., *Proceedings of the Fourth International Conference on Genetic Algorithms*, , Morgan Kaufmann, San Mateo, CA, 1991.

[8] G. Syswerda, *Uniform Crossover in Genetic Algorithms*, in Proceedings of the Third International Conference on Genetic Algorithms, J. D. Schaffer, ed., Morgan Kaufmann, San Mateo, CA, 1989, 2-9.

[9] J. D. Schaffer, R. A. Caruna, L. J. Eshelman, and R. Das, *A Study of Control Parameters Affecting Online Performance of Genetic Algorithms for Function Optimization*, in Proceedings of the Third International Conference on Genetic Algorithms, J. D. Schaffer, ed., Morgan Kaufmann, San Mateo, CA, 1989, 51-60.

[10] D. E. Goldberg, *Zen and the Art of Genetic Algorithms*, in Proceedings of the Third International Conference on Genetic Algorithms, J. D. Schaffer, ed., Morgan Kaufmann, San Mateo, CA, 1989, 80-85.

[11] K. A. De Jong, *An Analysis of the Behavior of a Class of Genetic Adaptive Systems*, Doctoral Thesis, University of Michigan, Ann Arbor, MI, 1975.

[12] P. L'Ecuyer, *Efficient and Portable Combined Random Number Generators*, Communications of the ACM, 31 (1988), 742-749, 774.

[13] M. Schollmeyer, *Noise Generators for Use in Computer Simulation*, M.S. Thesis, University of Missouri-Rolla, Rolla, MO, 1989.

[14] L. V. Edmondson, *Genetic Algorithms with 3-parent Crossover*, Ph.D. Dissertation, University of Missouri-Rolla, Rolla, MO, 1993.

III. A RELATIONSHIP BETWEEN THE METROPOLIS ALGORITHM AND THE TWO-MEMBERED EVOLUTION STRATEGY

A. INTRODUCTION

A significant amount of research has been done during the past two decades in the area of nature-inspired heuristic algorithms. These algorithms are designed to be robust problem-solving techniques which are typically applied to difficult optimization problems (such as those found in the class of problems labeled NP-complete). The two most common "natural" heuristic algorithms are simulated annealing and genetic algorithms. This paper briefly reviews the mechanics of the algorithms and then establishes a relationship between the Metropolis algorithm [1] from simulated annealing and a special form of a genetic algorithm known as the two-membered evolution strategy.

B. SIMULATED ANNEALING AND THE METROPOLIS ALGORITHM

Simulated annealing is modeled after the actual annealing process in condensed matter physics. In brief, annealing is the process in which the temperature of a solid in a heat bath is increased to a point at which the particles of the solid move freely with respect to one another, followed by a slow cooling of the heat bath. If the cooling is slow enough, then the particles line themselves up and reach a state with minimum energy.

If a system is in thermal equilibrium at a given temperature T, then its energy is probabilistically distributed among all different energy states E according to the Boltzmann probability distribution

$$Prob(E) \sim \exp\left(\frac{-E}{kT}\right)$$

where k is the Boltzmann constant. This means that, for any temperature T, there is a nonzero probability that the current local minimum is not the global minimum. The net effect of this is that the system can perform hillclimbing in an attempt to move from a local minimum to a better (possibly global) minimum [2,3].

The following pseudo-code form of the Metropolis algorithm incorporates the aforementioned hillclimbing strategy.

1. Generate a solution x_1 to the minimization problem and evaluate the objective function at x_1 to obtain E_1. ("Solution" simply means a valid answer to the problem and it does not imply optimality.)
2. Randomly perturb x_1 to obtain x_2 and evaluate the objective function at x_2 to obtain E_2.
3. Calculate the probability p that x_2 will become the incumbent solution.

   $$p = \exp\left[\frac{-(E_2 - E_1)}{kT}\right]$$

   If $p > 1$, then $p \leftarrow 1$.
4. Determine if x_2 will become the incumbent solution. Assume that random [0,1) generates a uniformly-distributed random number in the range [0,1).

```
        If $p >$ random [0,1) then $x_1 \leftarrow x_2$ and $E_1 \leftarrow E_2$.
5.      Determine if the algorithm should stop.
        If (termination criterion is not met) then goto step 2
                else stop with "optimal" solution $x_1$.
```

Examination of step 4 shows that the solution x_2 will always replace x_1 (and, hence, become the incumbent solution) whenever $E_2 \leq E_1$. This indicates that the solution at x_2 is better than the solution at x_1. There is also a chance that x_2 will replace x_1 as then incumbent solution when $E_2 > E_1$ (this is known as "hillclimbing").

Some possible termination criteria are having reached a maximum number of iterations or having successfully replaced the incumbent solution a maximum number of times. Clearly, these maximum numbers must be determined prior to the start of the algorithm.

For any particular invocation of the Metropolis algorithm, the temperature T maintains a constant value. The simulated annealing algorithm is a series of Metropolis algorithms with different (decreasing) values of T.

It is important to note, for the purposes of this paper, that the Metropolis algorithm always keeps a single solution as the incumbent. The perturbed solution will always unseat the incumbent if it is better, and it will sometimes unseat the incumbent if it is worse (this is hillclimbing).

C. GENETIC ALGORITHMS AND THE TWO-MEMBERED EVOLUTION STRATEGY

Genetic algorithms are randomized, population-based search procedures which utilize the Darwinian notion of "survival of the fittest." These algorithms were independently developed by Holland [4] at the University of Michigan and by Rechenberg and Schwefel [5] in Germany. The German versions are known as evolution strategies (ESs) and will be the focus of this section.

The general process of the two-membered ES, denoted (1+1)-ES, is to start with the single population member, mutate it (change it in some fashion prescribed by the mutation operator) to create a single offspring, and then select the better of the two to become the parent for the next generation. The "betterness" quality of an individual arises from the objective function evaluation. If the objective function is to be minimized, then the individual with the smallest function value becomes the parent.

Schwefel [6] describes the (1+1)-ES algorithm with the following 8-tuple:

$(1+1)\text{-ES} = (P^0, m, s, c_d, c_i, f, g, t)$

where

P^0	$=$	$(x^0, \sigma^0) \in I$	population, $I = \mathbb{R}^n \times \mathbb{R}^n$
m	$:$	$I \rightarrow I$	mutation operator
s	$:$	$I \times I \rightarrow I$	selection operator
$c_d, c_i \in \mathbb{R}$			step-size control
f	$:$	$\mathbb{R}^n \rightarrow \mathbb{R}$	objective function
g	$:$	$\mathbb{R}^n \rightarrow \mathbb{R}$	constraint functions

t : $I \times I \rightarrow \{0,1\}$ termination criterion

At any given time/generation r, P^r represents the parent and $m(P^r)$ is the child (mutated parent). Although the mutation operator can be generalized, it was originally defined in such a way that x'^r (the child) was the addition of the n-element vector x^r (the parent) and an n-element vector of independent, normally-distributed random numbers with zero mean and standard deviation σ^r. Assuming a minimization problem, the parent in generation $r+1$ would be the same as in generation r unless $f(x'^r) \leq f(x^r)$. The step-size controls were used to modify σ^r so that a successful mutation occurred approximately one-fifth of the time. The termination criterion could be defined in numerous ways, including the use of a maximum number of generations or a maximum CPU time.

Again, for the purposes of this paper, it is important to note that in the (1+1)-ES algorithm a single solution is maintained as the incumbent. This incumbent is perturbed each generation and then a selection operator chooses the incumbent for the next generation.

D. RELATIONSHIP BETWEEN THE METROPOLIS ALGORITHM AND (1+1)-ES

The following theorem establishes a relationship between the Metropolis algorithm and the (1+1)-ES algorithm.

Theorem. The Metropolis algorithm is a special case of the two-membered evolution strategy.

Proof. To prove this theorem, it is sufficient to show that the Metropolis algorithm can be defined with the same 8-tuple used for the (1+1)-ES algorithm.

Metropolis algorithm = $(P^0, m, s, c_d, c_i, f, g, t)$

P^0 represents the initial solution. In general, the value of σ^r is arbitrary (unless the mutation operator requires a standard deviation).

The Metropolis algorithm does not specify a particular perturbation method. Therefore, the mutation operator m can be defined in whatever manner is consistent with the perturbation method required by the specific instantiation of the Metropolis algorithm under consideration.

The selection operator s must be defined so that

$$P^{r+1} = \begin{cases} x'^r & \textit{if} \exp\left[\dfrac{-(f(x'^r)-f(x^r))}{kT}\right] > \textit{random}[0,1) \\ x^r & \textit{otherwise} \end{cases}$$

The values of c_d and c_i are arbitrary (unless σ^r needs to be modified so that the mutation success rate can be held approximately constant).

The choice of algorithm will have no impact on the objective function f or the constraint functions g. It is assumed that the mutation operator will generate perturbations that satisfy all constraint functions.

The Metropolis algorithm does not specify a particular termination criterion. Therefore, t can be defined in whatever manner is consistent with the termination criterion required by the specific instantiation of the Metropolis algorithm under consideration.

Remark. This theorem shows that, at a fundamental algorithmic level, the annealing process is a simplistic form of evolution.

E. EXAMPLE

Here is a simple example using the Metropolis algorithm. Suppose that the following distance matrix is given for the traveling salesperson problem.

city	A	B	C	D	E
A	-	5	9	2	12
B	5	-	6	11	4
C	9	6	-	7	9
D	2	11	7	-	11
E	12	4	9	11	-

Suppose that x_1 is the tour A-B-C-D-E. The associated objective function E_1 is 5+6+7+11+12=41. Now suppose that x_1 is perturbed by inverting the order of the second through fourth cities in the tour, yielding x_2 = A-D-C-B-E. The associated objective function E_2 is 2+7+6+4+12=31. Without loss of generality, assume that the Boltzmann parameters k and T are 1 and 0.99, respectively. Using step 3, $p \approx 24368$. Since the calculated value for p is greater than 1, it is reset to 1. Therefore, in step 4, x_2 becomes the incumbent solution.

Suppose that the next iteration perturbs the incumbent solution by inverting the order of the first and second cities, giving a tour of D-A-C-B-E with an objective function value of 33. Since $E_2 > E_1$, step 3 will yield a p value that is less than unity. Therefore, x_2 will replace x_1 as the incumbent solution only if p is greater than the random number generated in step 4. This process, known as hillclimbing, is used to allow the algorithm to escape from (possibly non-global) local minima.

The algorithm will terminate after either a predetermined number of iterations has been reached or after a predetermined number of successful reconfigurations has been reached.

Section D of this paper established that the (1+1)-ES is equivalent to the Metropolis algorithm when the parameters are chosen appropriately. Based on this equivalence, the (1+1)-ES would yield the same sequence of x-iterates as the Metropolis algorithm. Therefore, it is not necessary to repeat the example for the (1+1)-ES.

F. CONCLUSION

Randomized search techniques (including simulated annealing and genetic algorithms) have been applied to a wide variety of problems. Goldberg [7] lists genetic algorithm application problems from diverse disciplines such as biology, computer science, engineering, and social science. Aarts and van Laarhoven give a similar list for simulated annealing in [2].

A characteristic of many of these problems is that they are NP-complete. Although neither simulated annealing nor genetic algorithms can guarantee that an optimal solution to a problem will be found (especially for an NP-complete problem), they have been shown to be robust techniques that generally locate a near-optimal solution.

G. ACKNOWLEDGMENT

The author would like to thank an anonymous referee for providing suggestions to improve this paper.

REFERENCES

[1] N. Metropolis et al., "Equation of State Calculations by Fast Computing Machines," Journal of Chem. Physics, 21, 1953, 1087-1092.

[2] P. J. M. van Laarhoven and E. H. L. Aarts, Simulated Annealing: Theory and Applications, Kluwer Academic Publishers, 1987.

[3] W. H. Press et al., Numerical Recipes in C, Cambridge University Press, 1988.

[4] J. H. Holland, Adaptation in Natural and Artificial Systems, The University of Michigan Press, 1975.

[5] H. P. Schwefel, Numerical Optimization of Computer Models, John Wiley & Sons, 1981.

[6] H. P. Schwefel, "A Survey of Evolution Strategy," ed. R. K. Belew and L. B. Booker, Proceedings of the Fourth International Conference on Genetic Algorithms, Morgan Kaufmann, 1991, 2-9.

[7] D. E. Goldberg, Genetic Algorithms in Search, Optimization, and Machine Learning, Addison Wesley, 1989.

APPENDIX A
A Brief History of Genetic Algorithms

The most prevalent form of genetic algorithms (GAs) was developed by John Holland and his students at the University of Michigan in the late 1960's and early 1970's [1]. In true Darwinistic form, GAs have evolved to the point that many different genetic algorithm (GA) species exist. The biological analogy upon which GAs are based will break down if it is pushed to an extreme. In a similar manner, the (somewhat poetic) reference to the speciation of GAs is not intended to be mathematically precise. The idea of an "algorithmic species" is, at best, a fuzzy notion. However, the analogy does provide a useful framework within which the history of GAs can be explored.

Richard Dawkins [2] points out that biologists do not have a complete fossil record to use when investigating the development of species. Furthermore, even if it was available, its enormity would make its exhaustive study an intractable problem. The complete "fossil record" of GA research is available, but it is difficult to ascertain. The explosion of GA research during the past 20+ years makes its study a large (but tractable) undertaking.

The major events/results of GA research are summarized in this brief history. The reader should assume that only the major nodes and branches of the GA-research phylogenetic tree ("tree of life") are presented.

HOLLAND'S ORIGINAL MODEL

Holland is generally recognized as the Father of Genetic Algorithms. His contributions to the field are many and varied, with the most important being the firm root node that he provides to the GA phylogenetic tree. Specifically, his original GA model and its accompanying mathematical analysis provided a starting point for most other GA researchers to follow.

Holland's traditional, three-operator GA begins with a population of randomly-generated binary string creatures. This initial population is called Generation 1. The fitness of each individual in the population is evaluated using an objective function and then these objective function values are used to determine which individuals will be copied (or partially copied) into Generation 2.

This process of reproduction can be easily understood as a biased roulette wheel. Each individual is allocated an amount of the roulette wheel which is proportional to its objective function value. For example, suppose that there were six individuals in the population, numbered 1 through 6, and their respective fitnesses were 100, 200, 150, 400, 100, and 50. The sum of the fitness values is 1000, so individual number 1 would receive (100/1000)*100%=10% of the roulette wheel. Similarly, individuals 2 through 6 would receive 20, 15, 40, 10, and 5 percent, respectively. Individuals are then chosen for reproduction by spinning this weighted roulette wheel. This process is essentially the same as that described by Gillett [3] for the generation of simulation data.

Histograms of the cumulative distribution of the fitness values can be plotted with the x-axis representing individual population members and the y-axis ranging from 0 to 1. A uniformly-distributed pseudo-random number between 0 and 1 can be generated, plotted on the y-axis, projected horizontally until the cumulative distribution function or a discontinuity of this function is intersected, and then the corresponding individual can be read from the x-axis.

After individuals are selected for reproduction, the crossover and mutation operators are used to create offspring. Crossover is the most important of these two operators. Traditionally, this operator is used to mate two randomly-selected parents. Assuming that the length of the binary string creature is k, a uniformly-distributed pseudo-random integer value j is generated and serves as a crossover point. The first child is created by concatenating the bits in positions 1 through $(j-1)$ of the first parent with the bits in positions j through k of the second parent. Similarly, the second child receives bits 1 through $(j-1)$ from parent 2 and bits j through k from parent 1.

The mutation operator changes a bit from either 0 to 1 or 1 to 0. There is usually only a very small probability that mutation will occur (inversely proportional to the population size). The primary purpose of mutation is to ensure that there is a probability > 0 that diversity in the population will be maintained.

Under the assumption of generational replacement, the next generation is complete when n children are created (which is equivalent to $n/2$ matings). The n children become potential parents and their fitnesses are evaluated. The process is then repeated until a preset number of generations has been reached.

Table IV gives a simple example, adapted from Goldberg [4], illustrating the process. Suppose that the function $f(x) = x^2$ is to be maximized. If permissible values of x range from 0 to 15, inclusive, then they can be represented as binary strings of length 4. For simplicity, assume a small population size of 4. The initial population members are randomly generated.

Table IV. Simple GA Example - Generation 1

Member Number	x (base 2)	x (base 10)	f(x)	Prob. of selection
1	0111	7	49	0.165
2	1010	10	100	0.337
3	1100	12	144	0.485
4	0010	2	4	0.013
			$\Sigma = 297$	
			avg. = 74.25	

As seen in Table IV, the average fitness level is 74.25. Recall that the probability of selection for a given population member is obtained by dividing that member's f(x) value by the summation of the f(x) values for all population members.

Using the roulette-wheel selection process described above, assume that string 3 is chosen to mate with both string 1 and string 2. It is unlikely that string 4 would be chosen for mating because the probability of selection is so low (0.013). This is the mathematical analogy of the Darwinian notion of "survival of the fittest." Over the course of many generations, only the fittest population members will propagate.

Table V gives the mating pool, randomly determined crossover site, and the resulting new generation of binary strings. There are no mutations shown in this example because the probability of mutation is typically very low.

Table V. Simple GA Example - Generation 2

Parents	Crossover site	Next generation	x (base 10)	f(x)
1100	3	1101	13	169
0111	3	0110	6	36
1100	2	1110	14	196
1010	2	1000	8	64
				Σ = 465
				avg. = 116.25

Although this example is contrived, it illustrates the general GA process. The average fitness level has increased from 74.25 to 116.25 in a single generation. Inspection of the new population shows that strings 1 and 3 have a good chance of mating. Further inspection shows that there is potential for one of their offspring to be the optimal binary string of all 1's (15 in base 10).

In addition to the original GA model, Holland developed what has become known as the Fundamental Theorem of Genetic Algorithms. It is necessary to make an observation and to establish some definitions before examining this theorem.

The observation is that there are more items than specific strings being processed from generation to generation. At a more abstract level, similarity templates (schemata) are being processed. The GA is actually exploiting similarities between above-average strings. Schemata can be described for the binary alphabet using the notation standardized by Goldberg [4].

Given a binary string of length k and the wildcard symbol *, there are 3^k possible schemata. A schema is used as a pattern matching device. A specific string and schema match if they agree at every position (allowing for the * in the schema to match either 0 or 1 in the string). Goldberg provides an excellent description of schemata in [4].

Some definitions are required before stating the theorem. Let H be a schema with length k. The order of the schema is defined to be the number of fixed positions. It is denoted by $o(H)$ and can be calculated by counting the number of non-wildcard positions or, equivalently, by subtracting the number of wildcard positions from k.

The defining length of H is denoted by $\delta(H)$ and is the distance between the first and last specific string position in the schema. For example, H=0***1* has $\delta(H) = 4$.

The Fundamental Theorem of Genetic Algorithms, also known as the Schema Theorem, establishes a lower bound on the number of copies of a particular schema at time t+1, denoted m(H,t+1). Specifically,

$$m(H,t+1) \geq m(H,t) \frac{f(H)}{\bar{f}} \left[1 - p_c \frac{\delta(H)}{k-1} - o(H) p_m \right]$$

where $f(H)$ is the average fitness of strings representing schema H at time t, $\bar{f}$ is the average fitness of the entire population at time t, k is the length of H, p_c is the probability of crossover, and p_m is the probability of mutation.

This lower bound applies to a GA using the three operators of reproduction, crossover, and mutation (as described above). The specific details of the derivation of this theorem are in Goldberg [4].

The main pragmatic result of this theorem is that reproduction allocates exponentially increasing numbers of trials to above-average schemata. Similarly, exponentially decreasing numbers of trials are allocated to below-average schemata. This provides a mathematical foundation to the Darwinian notion of "survival of the fittest."

Holland [1] has shown that for each generation in which n population members are processed, $O(n^3)$ schemata are processed. This characteristic of GAs is known as implicit parallelism.

EVOLUTION STRATEGIES

At approximately the same time that Holland developed GAs, a set of techniques called evolution strategies (ESs) coevolved in Germany. ESs originated with Ingo Rechenberg and were further developed by Schwefel [5]. ESs were first applied to optimization problems with continuous parameters.

The first ES was a simple mutation-selection procedure with only two population members. The general process of this two-membered ES, denoted (1+1)-ES, is to start with the single population member, mutate it (change it in some fashion prescribed by the mutation operator) to create a single offspring, and then select the better of the two to become the parent for the next generation. The "betterness" quality of an individual arises from the objective function evaluation. If the objective function is to be maximized, then the individual with the largest function value becomes the parent.

This general process continues until some predetermined stopping criterion, such as reaching a maximum number of generations or reaching a maximum CPU time, is met. Schwefel [6] describes the (1+1)-ES algorithm with the following 8-tuple:

$$(1+1)\text{-ES} = (P^0, m, s, c_d, c_i, f, g, t)$$

where

P^0	$=$	$(x^0, \sigma^0) \in I$	population, $I = \mathbb{R}^n \times \mathbb{R}^n$
m	:	$I \to I$	mutation operator
s	:	$I \times I \to I$	selection operator
$c_d, c_i \in \mathbb{R}$			step-size control
f	:	$\mathbb{R}^n \to \mathbb{R}$	objective function
g	:	$\mathbb{R}^n \to \mathbb{R}$	constraint functions
t	:	$I \times I \to \{0,1\}$	termination criterion

It is interesting to observe that the (1+1)-ES is very similar to simulated annealing. In both methods, an individual is modified in some fashion, and then either the original individual or the modified individual is saved as the incumbent/best solution. Vincent Edmondson [7] has shown that, by appropriately choosing the mutation operator, selection operator, and termination criterion, the simulated annealing algorithm is a special case of the (1+1)-ES algorithm.

The (1+1)-ES algorithm has been generalized to the ($\mu+\lambda$)-ES and (μ,λ)-ES algorithms. In the ($\mu+\lambda$)-ES algorithm, there are μ population members in a given generation, from which λ children are produced. Generational replacement is not used, so it is possible for a very "fit" individual to survive for the entire duration of the execution of the algorithm.

The (μ,λ)-ES algorithm imposes the restriction of generational replacement. Therefore, each individual survives for only a single generation. As was observed with generational replacement in Holland's GA approach, this helps to reduce premature convergence. The risk, of course, is that a super individual will be lost/forgotten before the termination criterion is met.

These multimembered algorithms have tuple representations that are analogous to the 8-tuple representation of the (1+1)-ES algorithm given above. An overview of ESs can be found in [6], and a complete description is available in [5].

DE JONG AND THE PITT APPROACH

Ken De Jong, one of Holland's students, migrated to the University of Pittsburg after completing his seminal dissertation at the University of Michigan. Among his many contributions are the set of test functions for comparing GA performance, extensions to Holland's original model, the development of the Pitt approach, and his applications of GAs to NP-complete problems.

As noted by Goldberg [4], the set of test functions that De Jong developed for his dissertation included the following characteristics (clearly, not all of these occurred in a single test function): continuous/discontinuous, convex/nonconvex, unimodal/multimodal, quadratic/nonquadratic, low-dimensionality/high-dimensionality, and deterministic/stochastic. Specifically, the set of test functions can be found in Table VI.

Table VI. De Jong Test Suite

F1	$f_1(x_i) = \sum_{i=1}^{3} x_i^2,$	$-5.12 \leq x_i \leq 5.12$
F2	$f_2(x_i) = 100(x_1^2 - x_2)^2 + (1 - x_1)^2,$	$-2.048 \leq x_i \leq 2.048$
F3	$f_3(x_i) = \sum_{i=1}^{5} \text{integer}(x_i),$	$-5.12 \leq x_i \leq 5.12$
F4	$f_4(x_i) = \sum_{i=1}^{30} i x_i^4 + \text{Gauss}(0,1),$	$-1.28 \leq x_i \leq 1.28$
F5	$f_5(x_i) = 0.002 + \sum_{j=1}^{25} \frac{1}{j + \sum_{i=1}^{2} (x_i - a_{ij})^6}$	$-65.536 \leq x_i \leq 65.536$

De Jong [8] considered five extensions to Holland's original model, several of which provided the basis for further study for many GA researchers. A brief description of these extensions follows.

In the "elitist model," De Jong employed a godlike immortality operator to ensure that the best individual to date is always included in the current generation. Specifically, if x is the best individual developed up to generation t and the GA operators do not propagate x into generation t+1, then put x in generation t+1 anyway. This approach was found to work well on unimodal surfaces, but not on multimodal surfaces.

In the "expected-value model," De Jong attempted to reduce the stochastic errors that are inherent in roulette wheel selection by calculating the expected number of offspring for each individual in the population for a given generation t. Whenever an individual was selected for reproduction, the offspring count was reduced. An individual with an offspring count below zero was no longer eligible for reproduction in that generation. Overall, this turned out to be an improvement for all of the test functions.

The "elitist expected-value model" combined the previous two approaches. As with the "elitist model," it only worked well on unimodal surfaces.

The "crowding model" did away with the idea of generational replacement. Instead of generational replacement, it maintained a constant population size by employing a literal birth-death process. Whenever an individual was born, another population member was selected to die. Specifically, the individual chosen for termination was the one most similar to the newest population member. Resemblance was measured by using a bit-by-bit similarity count. This idea worked well for the (more difficult) multimodal functions.

The final extension was the "generalized crossover model." In this approach, the number of crossover points was treated as a parameter. Based on his limited experiments, De Jong concluded that more than one crossover point was not a good idea. Subsequent research [9,10,11] has shown that multiple crossover points can be used effectively.

Grefenstette [12] provides a succinct description of the development of both the "Michigan approach" and the "Pitt approach" to machine learning via GAs. In the "Michigan approach" a population consists of a single set of production rules. Each rule is assigned a strength based on its usefulness in obtaining an external payoff. The bucket brigade algorithm reallocates the strength according to the payoff actually received during problem solving.

In contrast, the population members in the "Pitt approach" are each a set of production rules. Instead of manipulating individual rules (as is done in the "Michigan approach"), the GA operators are applied to sets of production rules. Currently, researchers in both camps are participating in a friendly debate over which approach is best.

Some of De Jong's most recent work has been in the area of applying GAs to NP-complete problems [13]. One of the most difficult problems with GAs is in finding a population member representation that is amenable to GA operators. The subsequent discussion of the traveling salesperson problem will further clarify this problem.

The majority of GA theory assumes a binary coding scheme. One problem that naturally lends itself to a binary coding scheme is the SATISFIABILITY problem (commonly abbreviated as SAT). This was the first problem ever shown to be in the class of NP-complete problems (via Cook's Theorem and proof) [14].

One property of NP-complete problems is that there exists a polynomial-time transformation from any NP-complete problem to any other NP-complete problem. Specifically, Spears and De Jong [13] have applied GAs to SAT and other NP-complete problems that have been transformed (in polynomial-time) to instances of SAT. As expected, GAs are not competitive when compared with problem-specific algorithms, but the initial results show that GAs are effective, robust algorithms for the general class of NP-complete problems. Regrettably, this effectiveness does not mean that a polynomial-time algorithm has been found for SAT.

GOLDBERG

Another one of Holland's Ph.D. students who has become a significant contributor to GA research is David Goldberg. With a background in civil engineering, Goldberg's dissertation research involved the application of GAs to optimization and machine learning in natural gas pipeline control [15]. Interestingly (and somewhat atypically for an engineer), Goldberg's major contributions have been in the development and refinement of GA theory, and not in the application realm. From a pragmatic perspective, his most outstanding contribution to date has been his GA text [4]. It takes the reader from zero knowledge to GA state-of-the-art (circa 1989). Some of the most important theoretical contributions are summarized below.

The concepts of niche and speciation were incorporated into GAs and applied to multimodal function optimization [4,16,17]. If a multimodal function has more than one optimal or near-optimal solution, then genetic drift (stochastic errors in sampling caused by small population sizes) will cause the GA to converge to a single peak. Exploiting the notions of niche and speciation will allow proportionally-sized subpopulations to develop around different peaks. This is accomplished by forcing population members near a particular peak to share the fitness value (reward) at that peak. Holland [1] uses a two-armed bandit problem to illustrate the concept.

Another of Goldberg's theoretical contributions is the addition of dominance and diploidy to the GA [4,18]. The traditional GA used a haploid (single-stranded chromosome) representation which contained all relevant information. With a diploid (double-stranded chromosome) representation, each population member redundantly carries two strings of information, thereby requiring dominance operators to decode the strings and eliminate the conflict of redundancy. Essentially, this allows both "dominant" and "recessive" genes to be carried in the population. The net effect of this is long-term memory, since a recessive gene may be carried for many generations before becoming "active."

Another recent (published) contribution is the development of "messy GAs" by Goldberg, Deb, and Korb [19]. It is possible, with some deceptive problems, that the global solution will be bypassed because the representation of the population member is not tightly linked to the function. Messy GAs have variable-length population member representations. This allows important, tightly-coded substrings to be found and then treated as if the elements of the substring are permanently bound. These messy GAs appear both to reduce the "linkage problem" described above and to be most applicable to blind combinatorial problems.

ACKLEY AND SIGH

A particularly unique method was developed by David Ackley for his Ph.D. dissertation [20]. The approach, named stochastic iterated genetic hillclimbing (SIGH), is a population-based search strategy which uses a democratic society metaphor. The SIGH algorithm attempts to optimize an n-dimensional function by using a voting process to determine the bit value for each of the n dimensions. The result of the election is a single string with n characters (analogous to the government in a democratic society). Ackley assumes two political parties, "Plus" and "Minus." Both of the parties compete for each of the n positions in the contest. If the Plus party wins, then the position becomes a 1, and if the Minus party wins, then it becomes a 0. In the case of a tie, the winner is determined stochastically. Each iteration of the SIGH algorithm consists of an "election" phase, a "reaction" phase, and an "outcome" phase. During the election phase, a subset of the population votes for each of the n dimensions. For each election, every population member is classified as one of the following: "satisfied," "dissatisfied," or "apathetic." The only population members to participate in an election are those that are either satisfied or dissatisfied. Although apathetic population members do not vote in an election, it is possible that they might become either satisfied or dissatisfied and vote in a subsequent election.

During the reaction phase, all population members are compared to the winner of the election. The results of these comparisons determine the classification for each member. Specifically, members who, in a bit-by-bit comparison, closely match the winner are labeled "satisfied." Members who match at about half of the bits are labeled "apathetic," and members who match at only a very small number of bits are labeled "dissatisfied."

The election winner is evaluated by the objective function during the outcome phase. If the function value compares favorably to previous election winners, then satisfied voters receive the credit and dissatisfied voters receive the blame. The blame and credit allocations are reversed if the function value does not compare favorably. The election results provide a basis for the preferences of active (non-apathetic) voters to be adjusted.

Stochasticity plays two important roles in the SIGH algorithm. First, as described above, the winner of the election is randomly determined in the case of a tie vote. Second, voter reactions are stochastic and are based on the degree of match over mismatch between the voter and the election winner.

Complete details of the SIGH algorithm can be found in [20]. Succinct descriptions can be found in [21,22].

DAVIS AND HYBRIDIZATION EFFORTS

Most of the published GA researchers appear to be academicians who are interested in the robustness and general problem-solving capabilities of GAs. One distinct exception to this is Lawrence Davis. Although Davis has contributed to the advancement of GA theory, he is currently one of the strongest advocates for hybrid GAs. A partial motivation for this approach is capitalism. As stated in [23], Davis works for a consulting firm and optimizes for a living. His goal, instead of robustness, is to convince clients that GAs are the best algorithms for solving their problems. Since problem-specific algorithms generally outperform GAs, hybridization is a logical approach to take in pursuit of his goal. An overview of some of Davis' most significant contributions follows.

Coombs and Davis [24] developed an interesting approach to using GAs on a constrained optimization problem. Recognizing that some constraints can be very time-consuming to check, they labeled these as "Ice Age" constraints and only checked them every *k* generations (where *k* generations represents a length of time that is equivalent to an Ice Age).

In the same paper, Davis and Coombs also discuss the development and use of the LaMarck operator. Dawkins [2] describes LaMarckism as the (false) belief that acquired traits can be inherited by future generations. Although this notion is not biologically correct, it can be useful in a GA. If a population member does not represent a legal solution to the problem under consideration, then the LaMarck operator can be invoked to make it legal. The changes acquired through the LaMarck operator can then be inherited by future generations.

The conventional GA wisdom has been, since De Jong's seminal dissertation [8], to preset GA parameters. In [25], Davis considered an adaptive approach to setting these parameters. Although details of the method used can be found in the original paper, it is the motivation behind them that warrants observation. In accordance with the above comments regarding hybridization, Davis' motivation was to automate the process of finding appropriate parameter settings so that a given hybrid GA algorithm would perform well. Hybridization generally involves the addition of problem-specific operators. Without assistance in the process of setting parameters, it would be difficult to assess the quality of the hybrid algorithm.

The most pragmatic contribution to date from Davis is his book on the hybridization of GAs [23]. It contains clearly stated descriptions of GAs and methods to hybridize GAs. Although it does not contain much GA theory (that was obviously not Davis' intent), it is an excellent "how-to" book on GAs.

TRAVELING SALESPERSON PROBLEM

Thus far this history of GAs has presented the GA phylogenetic tree with some of the major GA researchers serving as nodes in the tree. As stated in the introductory paragraphs, the GA algorithm speciation is a fuzzy, imprecise notion. There are several other relevant issues in the GA research arena which need to be included and which do not logically fit into the phylogenetic tree structure described above. This section, dealing with the traveling salesperson problem (TSP), is the first of several covering these other relevant issues.

There are three main approaches to solving TSP with GAs. In no particular order, they are GAs with a reordering operator, GAs with a greedy crossover operator, and GAs with a genetic edge recombination operator. As briefly mentioned above, one of the major difficulties with the use of GAs to solve an instance of TSP is the representation of a population member. An example will illustrate the problem.

Suppose that a five-city TSP is represented in (the seemingly natural) permutation form. If each city is to be visited in the order that they are listed (with the assumption that the salesperson will travel from the last city listed back to the first city listed), then the following tours A and B are valid.

A = 1 3 2 5 4

B = 5 1 3 4 2

Applying the traditional GA crossover operator to A and B (with crossover sites at positions 2 and 4) will yield the following invalid tours labeled C and D.

C = 1 3 3 4 4

D = 5 1 2 5 2

It is clear that either the crossover operator or the representation of tours needs to be modified so that offspring will have the property of being a valid tour.

An example of a reordering operator that uses the permutation representation is partially matched crossover (PMX) [26]. Mechanistically, PMX takes two permutation strings (parents) and two uniformly selected crossover sites as input. The two crossover sites define a "matching section." String values inside the matching section are crossed between the parents via position-by-position exchanges. Positionwise exchanges are used to ensure valid tours.

Consider tours A and B from the five-city TSP described above. Assuming, once again, that the crossover sites are at 2 and 4, the following tours C and D would result from the application of PMX:

C = 1 2 3 4 5

D = 4 1 2 5 3

Specifically, after position-by-position exchange in the matching section, the 3 and the 2, and the 4 and the 5, exchange places.

It is important to note that a GA with the PMX operator works on a blind TSP. There is nothing in PMX which exploits any knowledge about the distance between any two cities. The selective pressure of the PMX operator comes only from the overall tour length.

Similar reordering operators (order crossover and cycle crossover) have been developed. Order crossover was developed by Derek Smith [27] and cycle crossover was developed by Davis [28]. Succinct descriptions of each of these reordering operators can be found in Goldberg [4].

The greedy crossover operator, developed by Grefenstette et al. [29], is a modified crossover operator which works on an adjacency representation of TSP tours. In an adjacency representation, a value of j in the i^{th} location implies that the salesperson goes from city i to city j. For example, the adjacency representation (3 1 5 2 4) indicates that the tour will go from city 1 to 3, from 3 to 5, from 5 to 4, from 4 to 2, and from 2 back to 1.

As with the permutation representation form described above, the application of the traditional GA crossover operator to strings with an adjacency representation can yield invalid tours. Therefore, a modified crossover operator was needed for the adjacency representation.

Mechanistically, the greedy crossover operator begins by randomly picking a starting city. The shorter edge of the two edges leaving the starting city in the parents is chosen, thereby determining the next city to visit. This process is continued until a complete tour is generated. If, during this process, inclusion of the shorter edge would create a cycle, then randomly choose an edge to extend the tour.

It is important to note that this operator, unlike the reordering operators described above, exploits the knowledge of the distance between specific cities. Accordingly, the greedy crossover is not applicable to the blind TSP.

The third TSP operator, the genetic edge recombination operator, was developed by Darrell Whitley et al. at Colorado State University [30]. The approach based on this operator tries to pass along information about the edges/links between cities by using an edge map. The edge map keeps track of all the connections from the parents that lead into and out of a city. Recall from above the five-city TSP tours labeled A and B.

A = 1 3 2 5 4

B = 5 1 3 4 2

The edge map for these two tours is:

city 1 has edges to/from 3, 4, and 5

city 2 has edges to/from 3, 4, and 5

city 3 has edges to/from 1, 2, and 4

city 4 has edges to/from 1, 2, 3, and 5

city 5 has edges to/from 1, 2, and 4

D'Ann Fuquay gives the following succinct description of the mechanics of the algorithm in [31].

> After construction of the edge lists, the offspring is generated as follows. Choose one of the parents at random and designate its first city as current city. To determine the *next* city, consult the current city's edge list. Select from this list the unused city which has the fewest entries in its own edge list. (If a tie occurs, make a random choice among tied cities.) The newly chosen next city becomes the current city and the process continues until the tour is completed. In light of the goal to pass along as many edges as possible, this selection method is important because it reduces the likelihood of leaving a city with an empty edge list. If this does happen however, the next city is chosen at random from the remaining unselected cities.

Again, it is important to note that this approach works without exploiting any information about the distance between specific cities. This characteristic makes the algorithm more robust.

PARALLEL GENETIC ALGORITHMS

The parallelization of GAs is a subject which has received some attention during the past few years. When one considers the biological analogy upon which GAs are based, it is immediately apparent that the reproduction process in GAs is inherently parallel. Although not the only possible parallelization, the following paragraph describes the main idea behind most parallel GA approaches.

Most approaches to the parallelization of GAs involve the allocation of subpopulations to different processors. Each processor then acts upon its subpopulation in traditional GA fashion. On occasion (such as once per generation), information about the fittest individual(s) is sent to neighboring processors. Each processor must then decide how to incorporate the new (potential) subpopulation members into the subpopulation. This process is repeated until some preset termination criterion is met. Original descriptions of this algorithm can be found in [32,33].

One interesting aspect of parallel GAs is that they accomplish speciation within the larger population. As in nature, when a population is geographically separated into subpopulations, it is quite probable that speciation will occur. However, the migration of the fittest individuals is not as biologically sound. Once two subpopulations have actually split into different species (this is the speciation process), it is no longer possible for any individual from one subpopulation to successfully mate with an individual from the other subpopulation.

It is clear that parallelization will continue to be a fertile area for GA research. Additional information about parallel GAs can be found in the parallel GA sections of the three most recent international GA conferences [34,35,36].

CONCLUSION

GAs have been applied to a wide variety of areas. Goldberg [4] provides an extensive list of GA applications ranging from engineering and computer science to the social sciences. Additional applications can be found in each of the proceedings from the international conferences on GAs [34,35,36,37]. It is anticipated that, because of their robust nature, GAs will continue to be applied to such diverse areas.

As stated above, the intent of this brief history of GAs was to present the major events/results of GA research. Accordingly, it was neither feasible nor desirable to list every GA researcher along with his/her contribution. The best sources for additional information are [4,23,34,35,36,37].

REFERENCES

[1] J. H. Holland, *Adaptation in Natural and Artificial Systems*, The University of Michigan Press, Ann Arbor, MI, 1975.

[2] R. Dawkins, *The Blind Watchmaker*, Norton, New York, NY, 1986.

[3] B. E. Gillett, *Introduction to Operations Research: A Computer-oriented Algorithmic Approach*, McGraw-Hill, New York, NY, 1976.

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

[5] H. Schwefel, *Numerical Optimization of Computer Models*, John Wiley & Sons, New York, NY, 1981.

[6] H. Schwefel, *A Survey of Evolution Strategies*, in Proceedings of the Fourth International Conference on Genetic Algorithms, R. K. Belew and L. B. Booker, eds., Morgan Kaufmann, San Mateo, CA, 1991, 2-9.

[7] L. V. Edmondson, *A Relationship Between the Metropolis Algorithm and the Two-Membered Evolution Strategy*, Missouri Journal of Mathematical Sciences, 5 (1993), 8-12.

[8] K. A. De Jong, *An Analysis of the Behavior of a Class of Genetic Adaptive Systems*, Doctoral Thesis, University of Michigan, Ann Arbor, MI, 1975.

[9] L. J. Eshelman, R. A. Caruana, and J. D. Schaffer, *Biases in the Crossover Landscape*, in Proceedings of the Third International Conference on Genetic Algorithms, J. D. Schaffer, ed., Morgan Kaufmann, San Mateo, CA, 1989, 10-19.

[10] J. D. Schaffer, R. A. Caruana, L. J. Eshelman, and R. Das, *A Study of Control Parameters Affecting Online Performance of Genetic Algorithms for Function Optimization*, in Proceedings of the Third International Conference on Genetic Algorithms, J. D. Schaffer, ed., Morgan Kaufmann, San Mateo, CA, 1989, 51-60.

[11] W. M. Spears and K. A. De Jong, *On the Virtues of Parameterized Uniform Crossover*, in Proceedings of the Fourth International Conference on Genetic Algorithms, R. K. Belew and L. B. Booker, eds., Morgan Kaufmann, San Mateo, CA, 1991, 230-236.

[12] J. J. Grefenstette, *Multilevel Credit Assignment in a Genetic Learning System*, in Genetic Algorithms and Their Applications: Proceedings of the Second International Conference on Genetic Algorithms, J. J. Grefenstette, ed., Lawrence Erlbaum Associates, Hillsdale, NJ, 1987, 202-209.

[13] K. A. De Jong and W. M. Spears, *Using Genetic Algorithms to Solve NP-Complete Problems*, in Proceedings of the Third International Conference on Genetic Algorithms, J. D. Schaffer, ed., Morgan Kaufmann, San Mateo, CA, 1989, 124-132.

[14] S. A. Cook, *The Complexity of Theorem-Proving Procedures*, in Proceedings of the Third Annual ACM Symposium on Theory of Computing, Association for Computing Machinery, New York, NY, 1971, 151-158.

[15] D. E. Goldberg, *Computer-Aided Gas Pipeline Operation Using Genetic Algorithms and Rule Learning*, Doctoral dissertation, University of Michigan, 1983.

[16] D. E. Goldberg and J. Richardson, *Genetic Algorithms with Sharing for Multimodal Function Optimization*, in Genetic Algorithms and Their Applications: Proceedings of the Second International Conference on Genetic Algorithms, J. J. Grefenstette, ed., Lawrence Erlbaum Associates, Hillsdale, NJ, 1987, 41-49.

[17] K. Deb and D. E. Goldberg, *An Investigation of Niche and Species Formation in Genetic Function Optimization*, in Proceedings of the Third International Conference on Genetic Algorithms, J. D. Schaffer, ed., Morgan Kaufmann, San Mateo, CA, 1989, 42-50.

[18] D. E. Goldberg and R. E. Smith, *Nonstationary Function Optimization Using Genetic Algorithms with Dominance and Diploidy*, in Genetic Algorithms and Their Applications: Proceedings of the Second International Conference on Genetic Algorithms, J. J. Grefenstette, ed., Lawrence Erlbaum Associates, Hillsdale, NJ, 1987, 59-68.

[19] D. E. Goldberg, K. Deb, and B. Korb, *Don't Worry, Be Messy*, in Proceedings of the Fourth International Conference on Genetic Algorithms, R. K. Belew and L. B. Booker, eds., Morgan Kaufmann, San Mateo, CA, 1991, 24-30.

[20] D. H. Ackley, *A Connectionist Machine for Genetic Hillclimbing*, Kluwer Academic Publishers, Boston, MA, 1987.

[21] D. H. Ackley, *An Empirical Study of Bit Vector Function Optimization*, in Genetic Algorithms and Simulated Annealing, L. Davis, ed., Morgan Kaufmann, San Mateo, CA, 1987, 170-204.

[22] D. H. Ackley, *A Connectionist Algorithm for Genetic Search*, in Proceedings of the First International Conference on Genetic Algorithms and Their Applications, J. J. Grefenstette, ed., Lawrence Erlbaum Associates, Hillsdale, NJ, 1985, 121-135.

[23] L. Davis, *Handbook of Genetic Algorithms*, Van Nostrand Reinhold, New York, NY, 1991.

[24] S. Coombs and L. Davis, *Genetic Algorithms and Communication Link Speed Design: Constraints and Operators*, in Genetic Algorithms and Their Applications: Proceedings of the Second International Conference on Genetic Algorithms, J. J. Grefenstette, ed., Lawrence Erlbaum Associates, Hillsdale, NJ, 1987, 257-260.

[25] L. Davis, *Adapting Operator Probabilities in Genetic Algorithms*, in Proceedings of the Third International Conference on Genetic Algorithms, J. D. Schaffer, ed., Morgan Kaufmann, San Mateo, CA, 1989, 61-69.

[26] D. E. Goldberg, *Alleles, Loci, and the Traveling Salesman Problem*, in Proceedings of the First International Conference on Genetic Algorithms and Their Applications, J. J. Grefenstette, ed., Lawrence Erlbaum Associates, Hillsdale, NJ, 1985, 154-159.

[27] D. Smith, *Bin Packing with Adaptive Search*, in Proceedings of the First International Conference on Genetic Algorithms and Their Applications, J. J. Grefenstette, ed., Lawrence Erlbaum Associates, Hillsdale, NJ, 1985, 202-207.

[28] L. Davis, *Job Shop Scheduling with Genetic Algorithms*, in Proceedings of the First International Conference on Genetic Algorithms and Their Applications, J. J. Grefenstette, ed., Lawrence Erlbaum Associates, Hillsdale, NJ, 1985, 136-140.

[29] J. J. Grefenstette, R. Gopal, B. Rosmaita, and D. Van Gucht, *Genetic Algorithms for the Traveling Salesman Problem*, in Proceedings of the First International Conference on Genetic Algorithms and Their Applications, J. J. Grefenstette, ed., Lawrence Erlbaum Associates, Hillsdale, NJ, 1985, 160-168.

[30] D. Whitley, T. Starkweather, and D. Fuquay, *Scheduling Problems and Traveling Salesmen: The Genetic Edge Recombination Operator*, in Proceedings of the Third International Conference on Genetic Algorithms, J. D. Schaffer, ed., Morgan Kaufmann, San Mateo, CA, 1989, 133-140.

[31] D. Fuquay, *Genetic Algorithm Solutions to the Traveling Salesman Problem - GENITOR and the Edge Recombination Operator*, Technical Report CS-89-112, Department of Computer Science, Colorado State University, 1989.

[32] C. Pettey, M. R. Lueze, and J. J. Grefenstette, *A Parallel Genetic Algorithm*, in Genetic Algorithms and Their Applications: Proceedings of the Second International Conference on Genetic Algorithms, J. J. Grefenstette, ed., Lawrence Erlbaum Associates, Hillsdale, NJ, 1987, 155-161.

[33] P. Jog and D. Van Gucht, *Parallelization of Probabilistic Sequential Search Algorithms*, in Genetic Algorithms and Their Applications: Proceedings of the Second International Conference on Genetic Algorithms, J. J. Grefenstette, ed., Lawrence Erlbaum Associates, Hillsdale, NJ, 1987, 170-176.

[34] J. J. Grefenstette, editor, *Genetic Algorithms and Their Applications: Proceedings of the Second International Conference on Genetic Algorithms*, Lawrence Erlbaum Associates, Hillsdale, NJ, 1987.

[35] J. D. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, Morgan Kaufmann, San Mateo, CA, 1989.

[36] R. K. Belew and L. B. Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, Morgan Kaufmann, San Mateo, CA, 1991.

[37] J. J. Grefenstette, editor, *Proceedings of the First International Conference on Genetic Algorithms and Their Applications*, Lawrence Erlbaum Associates, Hillsdale, NJ, 1985.

APPENDIX B
Detailed Uniform Crossover Results

TABLE VII. Uniform Crossover on *F1* With a Maximum of 50 Generations

Population Size	Probability of Crossover	2-Parent Average	3-Parent Average	Winner per Pop. Size
60	0.6	0.02847	0.03267	
60	0.7	0.02563	0.03575	
60	0.8	0.021795	0.030745	2-parent
60	0.9	0.03373	0.053575	
120	0.6	0.01551	0.01488	
120	0.7	0.015695	0.012725	
120	0.8	0.010175	0.014085	2-parent
120	0.9	0.01206	0.01697	
180	0.6	0.01252	0.01125	
180	0.7	0.010035	0.008315	
180	0.8	0.00835	0.007215	
180	0.9	0.007165	0.009785	2-parent
240	0.6	0.00513	0.00599	
240	0.7	0.00923	0.00491	3-parent
240	0.8	0.00878	0.007365	
240	0.9	0.00632	0.005375	

TABLE VIII. Uniform Crossover on *F1* With a Maximum of 100 Generations

Population Size	Probability of Crossover	2-Parent Average	3-Parent Average	Winner per Pop. Size
60	0.6	0.018575	0.01723	
60	0.7	0.026755	0.016115	
60	0.8	0.0103	0.010875	
60	0.9	0.00645	0.01987	2-parent
120	0.6	0.009075	0.00768	
120	0.7	0.006465	0.008015	
120	0.8	0.006525	0.006555	
120	0.9	0.0049	0.003835	3-parent
180	0.6	0.00552	0.00488	
180	0.7	0.00355	0.002795	
180	0.8	0.004475	0.002505	3-parent
180	0.9	0.00376	0.00317	
240	0.6	0.003725	0.002875	
240	0.7	0.001835	0.005875	
240	0.8	0.001855	0.003115	
240	0.9	0.001825	0.00194	2-parent

TABLE IX. Uniform Crossover on *F1* With a Maximum of 150 Generations

Population Size	Probability of Crossover	2-Parent Average	3-Parent Average	Winner per Pop. Size
60	0.6	0.008095	0.01206	
60	0.7	0.009895	0.01554	
60	0.8	0.006965	0.00914	2-parent
60	0.9	0.008985	0.008845	
120	0.6	0.00492	0.005055	
120	0.7	0.00469	0.00589	
120	0.8	0.004885	0.003395	
120	0.9	0.002255	0.00329	2-parent
180	0.6	0.00214	0.00247	
180	0.7	0.003115	0.00279	
180	0.8	0.002435	0.00265	
180	0.9	0.002385	0.001845	3-parent
240	0.6	0.002555	0.00179	
240	0.7	0.00191	0.00213	
240	0.8	0.001895	0.00223	
240	0.9	0.000795	0.00085	2-parent

TABLE X. Uniform Crossover on *F1* With a Maximum of 200 Generations

Population Size	Probability of Crossover	2-Parent Average	3-Parent Average	Winner per Pop. Size
60	0.6	0.008895	0.006895	
60	0.7	0.00579	0.013225	
60	0.8	0.00845	0.009105	
60	0.9	0.005775	0.008495	2-parent
120	0.6	0.00305	0.00445	
120	0.7	0.00327	0.005535	
120	0.8	0.003035	0.002415	
120	0.9	0.002235	0.002995	2-parent
180	0.6	0.00219	0.001925	
180	0.7	0.001835	0.00261	
180	0.8	0.00231	0.00182	
180	0.9	0.001215	0.001185	3-parent
240	0.6	0.002515	0.00161	
240	0.7	0.00134	0.00151	
240	0.8	0.001005	0.001675	
240	0.9	0.00079	0.001165	2-parent

TABLE XI. Uniform Crossover on *F2* With a Maximum of 50 Generations

Population Size	Probability of Crossover	2-Parent Average	3-Parent Average	Winner per Pop. Size
60	0.6	0.036118	0.03019	
60	0.7	0.036837	0.043424	
60	0.8	0.033571	0.020942	
60	0.9	0.027975	0.016662	3-parent
120	0.6	0.010424	0.015974	
120	0.7	0.019145	0.011375	
120	0.8	0.010086	0.011473	
120	0.9	0.007544	0.014936	2-parent
180	0.6	0.005161	0.007725	2-parent
180	0.7	0.008975	0.008729	
180	0.8	0.009118	0.005572	
180	0.9	0.006761	0.00528	
240	0.6	0.00509	0.007077	
240	0.7	0.007035	0.00433	
240	0.8	0.004478	0.003598	3-parent
240	0.9	0.005293	0.007366	

TABLE XII. Uniform Crossover on *F2* With a Maximum of 100 Generations

Population Size	Probability of Crossover	2-Parent Average	3-Parent Average	Winner per Pop. Size
60	0.6	0.02291	0.022105	
60	0.7	0.020452	0.006788	3-parent
60	0.8	0.017874	0.009629	
60	0.9	0.021226	0.016258	
120	0.6	0.005179	0.006882	
120	0.7	0.004897	0.006507	
120	0.8	0.005089	0.004505	
120	0.9	0.003973	0.007497	2-parent
180	0.6	0.003047	0.004072	
180	0.7	0.002775	0.004878	2-parent
180	0.8	0.003729	0.004029	
180	0.9	0.004286	0.003497	
240	0.6	0.002053	0.003945	
240	0.7	0.003448	0.003323	
240	0.8	0.003051	0.00186	3-parent
240	0.9	0.003842	0.002725	

TABLE XIII. Uniform Crossover on *F2* With a Maximum of 150 Generations

Population Size	Probability of Crossover	2-Parent Average	3-Parent Average	Winner per Pop. Size
60	0.6	0.009731	0.017465	
60	0.7	0.011003	0.008394	
60	0.8	0.006865	0.006694	
60	0.9	0.011672	0.006197	3-parent
120	0.6	0.00342	0.004123	
120	0.7	0.008124	0.004134	
120	0.8	0.004506	0.002752	3-parent
120	0.9	0.005703	0.003813	
180	0.6	0.001759	0.003327	
180	0.7	0.003031	0.003418	
180	0.8	0.003293	0.001644	
180	0.9	0.003003	0.001392	3-parent
240	0.6	0.002378	0.002295	
240	0.7	0.001803	0.00216	
240	0.8	0.001334	0.000986	3-parent
240	0.9	0.001226	0.001152	

TABLE XIV. Uniform Crossover on *F2* With a Maximum of 200 Generations

Population Size	Probability of Crossover	2-Parent Average	3-Parent Average	Winner per Pop. Size
60	0.6	0.008046	0.007321	
60	0.7	0.01251	0.007884	
60	0.8	0.011921	0.00407	3-parent
60	0.9	0.006319	0.00429	
120	0.6	0.004566	0.002962	
120	0.7	0.003353	0.002547	
120	0.8	0.003313	0.002412	3-parent
120	0.9	0.004338	0.002473	
180	0.6	0.001417	0.002916	
180	0.7	0.001921	0.002555	
180	0.8	0.002174	0.001501	
180	0.9	0.001731	0.000859	3-parent
240	0.6	0.001397	0.00127	
240	0.7	0.001704	0.001104	
240	0.8	0.001465	0.000963	3-parent
240	0.9	0.001499	0.001025	

TABLE XV. Uniform Crossover on *F3* With a Maximum of 50 Generations

Population Size	Probability of Crossover	2-Parent Average	3-Parent Average	Winner per Pop. Size
60	0.6	4.35	4.1	
60	0.7	3.5	4.0	
60	0.8	3.35	5.45	2-parent
60	0.9	4.1	3.8	
120	0.6	2.0	1.2	
120	0.7	1.4	0.75	3-parent
120	0.8	2.3	1.15	
120	0.9	1.9	1.2	
180	0.6	1.0	1.3	
180	0.7	0.8	0.5	
180	0.8	1.15	0.95	
180	0.9	0.2	0.85	2-parent
240	0.6	0.5	0.7	
240	0.7	0.15	0.4	
240	0.8	0.4	0.8	
240	0.9	0.05	0.55	2-parent

TABLE XVI. Uniform Crossover on *F3* With a Maximum of 100 Generations

Population Size	Probability of Crossover	2-Parent Average	3-Parent Average	Winner per Pop. Size
60	0.6	4.6	4.95	
60	0.7	3.7	3.8	
60	0.8	3.35	4.7	2-parent
60	0.9	4.95	3.6	
120	0.6	1.65	1.3	
120	0.7	1.7	1.1	
120	0.8	1.7	1.35	
120	0.9	1.6	1.05	3-parent
180	0.6	0.85	1.25	
180	0.7	0.6	0.6	
180	0.8	0.5	0.5	
180	0.9	0.25	0.65	2-parent
240	0.6	0.4	1.0	
240	0.7	0.35	0.5	
240	0.8	0.45	0.7	
240	0.9	0.0	0.6	2-parent

TABLE XVII. Uniform Crossover on *F3* With a Maximum of 150 Generations

Population Size	Probability of Crossover	2-Parent Average	3-Parent Average	Winner per Pop. Size
60	0.6	4.75	4.45	
60	0.7	3.7	4.4	2-parent
60	0.8	4.7	3.8	
60	0.9	4.35	4.0	
120	0.6	2.45	2.35	
120	0.7	2.25	1.2	
120	0.8	1.9	1.9	
120	0.9	1.5	0.95	3-parent
180	0.6	0.95	1.35	
180	0.7	0.5	1.0	
180	0.8	1.15	0.4	
180	0.9	0.15	0.65	2-parent
240	0.6	0.95	0.6	
240	0.7	0.8	0.4	
240	0.8	0.55	0.4	
240	0.9	0.0	0.55	2-parent

TABLE XVIII. Uniform Crossover on *F3* With a Maximum of 200 Generations

Population Size	Probability of Crossover	2-Parent Average	3-Parent Average	Winner per Pop. Size
60	0.6	5.35	4.6	
60	0.7	3.9	3.9	
60	0.8	2.9	4.2	2-parent
60	0.9	4.05	3.5	
120	0.6	2.1	2.4	
120	0.7	1.9	1.0	3-parent
120	0.8	1.45	1.75	
120	0.9	1.45	1.35	
180	0.6	1.1	1.45	
180	0.7	0.5	0.6	
180	0.8	0.6	0.35	3-parent
180	0.9	0.45	0.85	
240	0.6	0.55	0.5	
240	0.7	0.35	0.25	
240	0.8	0.2	0.45	
240	0.9	0.15	0.8	2-parent

TABLE XIX. Uniform Crossover on *F4* With a Maximum of 50 Generations

Population Size	Probability of Crossover	2-Parent Average	3-Parent Average	Winner per Pop. Size
60	0.6	47.35043	46.1915	
60	0.7	43.75472	47.55958	
60	0.8	38.58237	44.08317	2-parent
60	0.9	43.517	44.45015	
120	0.6	41.72298	41.5537	
120	0.7	39.07043	44.28386	
120	0.8	39.07889	40.51947	
120	0.9	38.60396	37.01249	3-parent
180	0.6	35.38194	36.8028	
180	0.7	37.67159	40.58131	
180	0.8	34.66783	37.46107	2-parent
180	0.9	36.62501	35.07707	
240	0.6	34.85803	35.88494	
240	0.7	33.77187	37.68944	
240	0.8	30.80801	38.24512	2-parent
240	0.9	33.06545	33.86475	

TABLE XX. Uniform Crossover on *F4* With a Maximum of 100 Generations

Population Size	Probability of Crossover	2-Parent Average	3-Parent Average	Winner per Pop. Size
60	0.6	7.460341	10.97718	
60	0.7	7.776963	5.001844	3-parent
60	0.8	8.027873	8.372672	
60	0.9	6.940345	6.9928	
120	0.6	9.360252	8.159231	3-parent
120	0.7	10.97313	12.27861	
120	0.8	10.72019	10.24858	
120	0.9	11.82037	10.61832	
180	0.6	12.22453	13.45867	
180	0.7	12.58127	13.81281	
180	0.8	12.39723	12.02795	
180	0.9	13.50581	11.87605	3-parent
240	0.6	13.94234	14.92002	
240	0.7	16.05545	15.96831	
240	0.8	14.09105	15.06074	
240	0.9	11.79601	13.3276	2-parent

TABLE XXI. Uniform Crossover on *F4* With a Maximum of 150 Generations

Population Size	Probability of Crossover	2-Parent Average	3-Parent Average	Winner per Pop. Size
60	0.6	4.789884	7.322733	
60	0.7	4.012892	3.60404	
60	0.8	4.287168	4.588309	
60	0.9	3.563293	3.629694	2-parent
120	0.6	4.349525	4.830629	
120	0.7	4.956127	4.22348	
120	0.8	3.938178	4.726036	2-parent
120	0.9	5.121316	4.559636	
180	0.6	6.564323	5.97817	
180	0.7	4.956511	6.117572	2-parent
180	0.8	5.230356	5.478578	
180	0.9	6.100008	5.105107	
240	0.6	6.220529	6.560764	
240	0.7	6.328065	5.67227	3-parent
240	0.8	5.930705	6.299047	
240	0.9	6.636116	6.633134	

TABLE XXII. Uniform Crossover on *F4* With a Maximum of 200 Generations

Population Size	Probability of Crossover	2-Parent Average	3-Parent Average	Winner per Pop. Size
60	0.6	3.602702	6.902592	
60	0.7	3.406705	3.416652	
60	0.8	3.237801	3.075595	3-parent
60	0.9	3.60196	3.341457	
120	0.6	3.290693	3.846759	2-parent
120	0.7	3.580497	3.337076	
120	0.8	4.152585	3.577069	
120	0.9	3.929103	3.666892	
180	0.6	3.636469	3.976383	
180	0.7	3.675861	3.802531	
180	0.8	4.262177	2.922713	3-parent
180	0.9	3.907649	4.478501	
240	0.6	4.219348	4.213084	
240	0.7	5.07175	4.721648	
240	0.8	4.621524	4.2732	
240	0.9	4.050683	4.382298	2-parent

TABLE XXIII. Uniform Crossover on *F5* With a Maximum of 50 Generations

Population Size	Probability of Crossover	2-Parent Average	3-Parent Average	Winner per Pop. Size
60	0.6	0.00200768132	0.00200768323	
60	0.7	0.00200767551	0.00200767399	3-parent
60	0.8	0.00200768289	0.00200767441	
60	0.9	0.00200768148	0.00200768048	
120	0.6	0.00200767075	0.00200766826	
120	0.7	0.00200766637	0.00200766685	
120	0.8	0.00200766683	0.00200766385	3-parent
120	0.9	0.00200766737	0.00200766908	
180	0.6	0.0020076694	0.00200766441	
180	0.7	0.00200766429	0.00200766265	
180	0.8	0.00200766484	0.00200766151	3-parent
180	0.9	0.00200766546	0.00200766287	
240	0.6	0.00200766388	0.00200766257	
240	0.7	0.00200766204	0.0020076641	
240	0.8	0.00200766067	0.0020076618	
240	0.9	0.00200766035	0.00200766233	2-parent

TABLE XXIV. Uniform Crossover on *F5* With a Maximum of 100 Generations

Population Size	Probability of Crossover	2-Parent Average	3-Parent Average	Winner per Pop. Size
60	0.6	0.00200768013	0.00200766754	3-parent
60	0.7	0.00200766892	0.00200766892	
60	0.8	0.00200767609	0.00200766824	
60	0.9	0.00200767181	0.00200766977	
120	0.6	0.00200766867	0.0020076657	
120	0.7	0.00200766373	0.0020076611	3-parent
120	0.8	0.00200766132	0.00200766197	
120	0.9	0.00200766371	0.00200766529	
180	0.6	0.00200766567	0.00200766403	
180	0.7	0.00200766082	0.00200765931	3-parent
180	0.8	0.00200766024	0.00200766106	
180	0.9	0.00200766181	0.00200766056	
240	0.6	0.00200766332	0.00200766008	
240	0.7	0.00200765844	0.00200765839	
240	0.8	0.00200766073	0.00200765809	
240	0.9	0.00200765707	0.00200766002	2-parent

TABLE XXV. Uniform Crossover on *F5* With a Maximum of 150 Generations

Population Size	Probability of Crossover	2-Parent Average	3-Parent Average	Winner per Pop. Size
60	0.6	0.00200767759	0.00200766626	
60	0.7	0.00200766711	0.00200766916	
60	0.8	0.00200766991	0.00200766216	3-parent
60	0.9	0.00200766562	0.00200766517	
120	0.6	0.00200766645	0.00200766181	
120	0.7	0.00200766207	0.00200766149	
120	0.8	0.0020076616	0.00200765918	
120	0.9	0.00200765833	0.00200766636	2-parent
180	0.6	0.00200766816	0.00200766107	
180	0.7	0.00200765986	0.00200766014	
180	0.8	0.00200765827	0.0020076577	3-parent
180	0.9	0.00200766153	0.00200765901	
240	0.6	0.00200766541	0.00200765929	
240	0.7	0.00200765738	0.00200765812	
240	0.8	0.00200765754	0.00200765726	
240	0.9	0.0020076568	0.00200765726	2-parent

TABLE XXVI. Uniform Crossover on *F5* With a Maximum of 200 Generations

Population Size	Probability of Crossover	2-Parent Average	3-Parent Average	Winner per Pop. Size
60	0.6	0.00200767489	0.0020076627	3-parent
60	0.7	0.00200766425	0.00200767027	
60	0.8	0.00200766395	0.00200767556	
60	0.9	0.00200766522	0.00200766598	
120	0.6	0.00200766457	0.00200765989	
120	0.7	0.00200766132	0.00200765934	
120	0.8	0.00200765949	0.00200765615	3-parent
120	0.9	0.00200766041	0.00200766099	
180	0.6	0.00200766458	0.00200765893	
180	0.7	0.00200765787	0.00200765818	
180	0.8	0.00200765747	0.00200765734	3-parent
180	0.9	0.00200765874	0.00200765755	
240	0.6	0.00200766399	0.00200765581	
240	0.7	0.00200765647	0.00200765674	
240	0.8	0.00200765685	0.00200765578	3-parent
240	0.9	0.00200765587	0.0020076581	

APPENDIX C

Detailed Traditional Crossover Results

TABLE XXVII. Traditional Crossover on *F1* With a Maximum of 50 Generations
(3-parent approach using random 3 out of 6 children)

Population Size	Probability of Crossover	2-Parent Average	3-Parent Average	Winner per Pop. Size
60	0.6	0.09095	0.0999	
60	0.7	0.101505	0.06959	3-parent
60	0.8	0.0784	0.134395	
60	0.9	0.083005	0.070675	
120	0.6	0.0485	0.041375	
120	0.7	0.050485	0.036675	
120	0.8	0.03289	0.04358	
120	0.9	0.03967	0.01641	3-parent
180	0.6	0.02842	0.030325	
180	0.7	0.02916	0.026055	
180	0.8	0.02116	0.018525	
180	0.9	0.025055	0.014235	3-parent
240	0.6	0.020545	0.024085	
240	0.7	0.01673	0.01391	
240	0.8	0.01901	0.029175	
240	0.9	0.022335	0.01298	3-parent

TABLE XXVIII. Traditional Crossover on *F1* With a Maximum of 100 Generations

(3-parent approach using random 3 out of 6 children)

Population Size	Probability of Crossover	2-Parent Average	3-Parent Average	Winner per Pop. Size
60	0.6	0.045145	0.103325	
60	0.7	0.03248	0.09695	
60	0.8	0.01889	0.04904	2-parent
60	0.9	0.047945	0.04209	
120	0.6	0.03599	0.012485	
120	0.7	0.011815	0.007055	3-parent
120	0.8	0.01224	0.015145	
120	0.9	0.01824	0.01245	
180	0.6	0.0186	0.009875	
180	0.7	0.011185	0.003495	3-parent
180	0.8	0.011355	0.015605	
180	0.9	0.008815	0.00983	
240	0.6	0.016675	0.0102	
240	0.7	0.00562	0.00573	2-parent
240	0.8	0.00593	0.012165	
240	0.9	0.00719	0.00773	

TABLE XXIX. Traditional Crossover on *F1* With a Maximum of 150 Generations

(3-parent approach using random 3 out of 6 children)

Population Size	Probability of Crossover	2-Parent Average	3-Parent Average	Winner per Pop. Size
60	0.6	0.03494	0.025485	
60	0.7	0.016225	0.020145	
60	0.8	0.015935	0.02093	2-parent
60	0.9	0.07179	0.02508	
120	0.6	0.020935	0.00574	3-parent
120	0.7	0.010565	0.01288	
120	0.8	0.00863	0.017635	
120	0.9	0.020415	0.008495	
180	0.6	0.01292	0.00685	
180	0.7	0.00757	0.00414	3-parent
180	0.8	0.007345	0.00517	
180	0.9	0.00834	0.008805	
240	0.6	0.00669	0.00443	
240	0.7	0.00434	0.00388	
240	0.8	0.00342	0.006605	2-parent
240	0.9	0.005545	0.004205	

TABLE XXX. Traditional Crossover on *F1* With a Maximum of 200 Generations

(3-parent approach using random 3 out of 6 children)

Population Size	Probability of Crossover	2-Parent Average	3-Parent Average	Winner per Pop. Size
60	0.6	0.02378	0.03552	
60	0.7	0.010925	0.011045	
60	0.8	0.00661	0.01944	
60	0.9	0.02964	0.00508	3-parent
120	0.6	0.01722	0.010145	
120	0.7	0.00694	0.004775	3-parent
120	0.8	0.00663	0.011335	
120	0.9	0.012235	0.00728	
180	0.6	0.018	0.013675	
180	0.7	0.006645	0.002805	3-parent
180	0.8	0.005935	0.009015	
180	0.9	0.00489	0.00433	
240	0.6	0.00064	0.002185	2-parent
240	0.7	0.004055	0.00334	
240	0.8	0.00333	0.007545	
240	0.9	0.00318	0.00339	

TABLE XXXI. Traditional Crossover on *F2* With a Maximum of 50 Generations (3-parent approach using random 3 out of 6 children)

Population Size	Probability of Crossover	2-Parent Average	3-Parent Average	Winner per Pop. Size
60	0.6	0.043925	0.051092	
60	0.7	0.03957	0.086503	
60	0.8	0.035831	0.057163	
60	0.9	0.048861	0.033873	3-parent
120	0.6	0.036901	0.015334	
120	0.7	0.020055	0.015495	
120	0.8	0.009347	0.018708	2-parent
120	0.9	0.0152	0.012462	
180	0.6	0.016152	0.015412	
180	0.7	0.011937	0.00811	
180	0.8	0.008612	0.008086	
180	0.9	0.012575	0.006201	3-parent
240	0.6	0.008243	0.006007	
240	0.7	0.005415	0.005327	
240	0.8	0.008889	0.004963	
240	0.9	0.013365	0.004713	3-parent

TABLE XXXII. Traditional Crossover on *F2* With a Maximum of 100 Generations

(3-parent approach using random 3 out of 6 children)

Population Size	Probability of Crossover	2-Parent Average	3-Parent Average	Winner per Pop. Size
60	0.6	0.020596	0.021296	
60	0.7	0.027332	0.02855	
60	0.8	0.018699	0.052397	2-parent
60	0.9	0.032807	0.023874	
120	0.6	0.00973	0.01154	
120	0.7	0.013187	0.013827	
120	0.8	0.005214	0.013648	2-parent
120	0.9	0.011156	0.005938	
180	0.6	0.013082	0.007731	
180	0.7	0.007625	0.006177	
180	0.8	0.004441	0.005358	
180	0.9	0.006742	0.004351	3-parent
240	0.6	0.005715	0.003262	
240	0.7	0.003051	0.002962	3-parent
240	0.8	0.003141	0.00383	
240	0.9	0.004655	0.003026	

TABLE XXXIII. Traditional Crossover on *F2* With a Maximum of 150 Generations

(3-parent approach using random 3 out of 6 children)

Population Size	Probability of Crossover	2-Parent Average	3-Parent Average	Winner per Pop. Size
60	0.6	0.018237	0.029235	
60	0.7	0.009509	0.021003	
60	0.8	0.011167	0.042582	
60	0.9	0.019174	0.008503	3-parent
120	0.6	0.0127	0.004934	
120	0.7	0.00755	0.007735	
120	0.8	0.003679	0.010479	2-parent
120	0.9	0.005656	0.005603	
180	0.6	0.004709	0.004605	
180	0.7	0.003797	0.005031	
180	0.8	0.003123	0.003469	
180	0.9	0.003994	0.002392	3-parent
240	0.6	0.002761	0.003335	
240	0.7	0.002202	0.001934	
240	0.8	0.003052	0.004297	
240	0.9	0.002302	0.001873	3-parent

TABLE XXXIV. Traditional Crossover on *F2* With a Maximum of 200 Generations

(3-parent approach using random 3 out of 6 children)

Population Size	Probability of Crossover	2-Parent Average	3-Parent Average	Winner per Pop. Size
60	0.6	0.021105	0.015963	
60	0.7	0.017838	0.019395	
60	0.8	0.010575	0.01681	2-parent
60	0.9	0.018035	0.015401	
120	0.6	0.009948	0.00792	
120	0.7	0.003948	0.010945	
120	0.8	0.003508	0.005088	
120	0.9	0.003038	0.002856	3-parent
180	0.6	0.004197	0.004232	
180	0.7	0.00334	0.002426	
180	0.8	0.002857	0.005127	
180	0.9	0.002693	0.00165	3-parent
240	0.6	0.003387	0.003483	
240	0.7	0.002148	0.002443	
240	0.8	0.001449	0.002944	
240	0.9	0.00216	0.000857	3-parent

TABLE XXXV. Traditional Crossover on *F3* With a Maximum of 50 Generations (3-parent approach using random 3 out of 6 children)

Population Size	Probability of Crossover	2-Parent Average	3-Parent Average	Winner per Pop. Size
60	0.6	7.65	6.25	
60	0.7	5.95	6.95	
60	0.8	6.6	6.05	
60	0.9	5.5	5.85	2-parent
120	0.6	5.4	3.6	
120	0.7	5.05	2.25	3-parent
120	0.8	3.65	2.4	
120	0.9	2.65	2.55	
180	0.6	3.05	4.3	
180	0.7	3.3	1.45	
180	0.8	3.2	1.75	
180	0.9	2.7	0.6	3-parent
240	0.6	2.7	5.75	
240	0.7	3.2	0.8	
240	0.8	2.25	0.75	
240	0.9	1.5	0.35	3-parent

TABLE XXXVI. Traditional Crossover on *F3* With a Maximum of 100 Generations

(3-parent approach using random 3 out of 6 children)

Population Size	Probability of Crossover	2-Parent Average	3-Parent Average	Winner per Pop. Size
60	0.6	6.8	5.25	
60	0.7	6.95	5.35	
60	0.8	7.6	4.3	3-parent
60	0.9	5.25	4.8	
120	0.6	4.75	4.25	
120	0.7	4.55	2.75	
120	0.8	3.05	2.45	
120	0.9	3.75	2.15	3-parent
180	0.6	3.0	4.15	
180	0.7	3.6	1.2	3-parent
180	0.8	2.65	1.75	
180	0.9	2.85	1.9	
240	0.6	3.15	4.8	
240	0.7	3.05	1.45	
240	0.8	1.5	1.2	
240	0.9	1.85	0.65	3-parent

TABLE XXXVII. Traditional Crossover on *F3* With a Maximum of 150 Generations

(3-parent approach using random 3 out of 6 children)

Population Size	Probability of Crossover	2-Parent Average	3-Parent Average	Winner per Pop. Size
60	0.6	7.6	6.5	
60	0.7	6.35	5.0	
60	0.8	6.55	5.4	
60	0.9	6.5	4.45	3-parent
120	0.6	5.5	4.7	
120	0.7	4.35	1.9	3-parent
120	0.8	3.95	3.2	
120	0.9	3.1	2.7	
180	0.6	2.05	4.25	
180	0.7	1.0	1.7	2-parent
180	0.8	1.2	2.5	
180	0.9	1.1	2.3	
240	0.6	3.2	4.05	
240	0.7	2.55	1.05	
240	0.8	1.9	1.35	
240	0.9	1.9	0.25	3-parent

TABLE XXXVIII. Traditional Crossover on *F3* With a Maximum of 200 Generations

(3-parent approach using random 3 out of 6 children)

Population Size	Probability of Crossover	2-Parent Average	3-Parent Average	Winner per Pop. Size
60	0.6	6.75	7.0	
60	0.7	5.8	4.35	
60	0.8	6.5	2.85	3-parent
60	0.9	6.1	4.4	
120	0.6	5.45	4.1	
120	0.7	3.6	3.05	
120	0.8	3.9	2.65	
120	0.9	3.0	2.0	3-parent
180	0.6	3.05	4.3	
180	0.7	3.3	1.75	
180	0.8	3.05	1.85	
180	0.9	2.0	1.4	3-parent
240	0.6	3.75	3.95	
240	0.7	2.45	1.5	
240	0.8	2.55	1.4	
240	0.9	1.65	0.85	3-parent

TABLE XXXIX. Traditional Crossover on *F4* With a Maximum of 50 Generations

(3-parent approach using random 3 out of 6 children)

Population Size	Probability of Crossover	2-Parent Average	3-Parent Average	Winner per Pop. Size
60	0.6	52.32811	50.4896	
60	0.7	50.2521	43.19021	3-parent
60	0.8	45.99219	47.79633	
60	0.9	52.07662	48.2049	
120	0.6	44.01746	41.98819	
120	0.7	46.01452	42.34553	
120	0.8	40.77805	44.14646	2-parent
120	0.9	44.93026	45.83559	
180	0.6	44.84551	37.9266	3-parent
180	0.7	43.20634	42.13426	
180	0.8	44.52702	42.94075	
180	0.9	43.57111	40.86885	
240	0.6	39.72294	43.62703	
240	0.7	43.41834	40.88328	
240	0.8	41.89307	38.87868	
240	0.9	42.98437	38.46299	3-parent

TABLE XL. Traditional Crossover on *F4* With a Maximum of 100 Generations
(3-parent approach using random 3 out of 6 children)

Population Size	Probability of Crossover	2-Parent Average	3-Parent Average	Winner per Pop. Size
60	0.6	32.21346	24.97913	
60	0.7	16.11699	25.33706	2-parent
60	0.8	19.38805	50.10294	
60	0.9	32.10905	51.79087	
120	0.6	13.40055	12.63022	
120	0.7	12.5976	12.35693	
120	0.8	10.67787	10.1048	
120	0.9	9.077796	10.36711	2-parent
180	0.6	13.7977	16.91911	
180	0.7	14.3492	14.62608	
180	0.8	13.53767	13.80741	
180	0.9	12.6264	14.23888	2-parent
240	0.6	16.89635	5.456462	3-parent
240	0.7	13.84845	15.40694	
240	0.8	13.91625	14.1576	
240	0.9	14.74651	7.525496	

TABLE XLI. Traditional Crossover on *F4* With a Maximum of 150 Generations (3-parent approach using random 3 out of 6 children)

Population Size	Probability of Crossover	2-Parent Average	3-Parent Average	Winner per Pop. Size
60	0.6	26.72981	22.06842	
60	0.7	34.19218	16.41104	
60	0.8	15.23151	22.66009	2-parent
60	0.9	21.90235	15.50279	
120	0.6	7.607474	5.099983	
120	0.7	5.559006	5.974291	
120	0.8	6.025749	5.016747	
120	0.9	5.672263	4.651987	3-parent
180	0.6	5.126395	7.603948	2-parent
180	0.7	5.499776	5.874581	
180	0.8	6.582014	5.335593	
180	0.9	6.012601	5.999213	
240	0.6	6.005873	4.183078	3-parent
240	0.7	6.662924	6.613931	
240	0.8	6.308006	5.310423	
240	0.9	6.654461	6.687661	

TABLE XLII. Traditional Crossover on *F4* With a Maximum of 200 Generations

(3-parent approach using random 3 out of 6 children)

Population Size	Probability of Crossover	2-Parent Average	3-Parent Average	Winner per Pop. Size
60	0.6	24.83604	20.73008	
60	0.7	25.71368	19.428	
60	0.8	19.29439	34.6663	
60	0.9	15.29822	19.42533	2-parent
120	0.6	3.419861	5.509111	2-parent
120	0.7	6.471318	4.643054	
120	0.8	5.009745	3.993501	
120	0.9	3.706555	4.261894	
180	0.6	4.854252	5.17658	
180	0.7	4.201519	4.03831	
180	0.8	5.062804	3.69541	3-parent
180	0.9	4.755725	3.966435	
240	0.6	4.508327	3.959367	3-parent
240	0.7	4.951602	4.475498	
240	0.8	5.353422	4.043203	
240	0.9	4.745058	4.81143	

TABLE XLIII. Traditional Crossover on *F5* With a Maximum of 50 Generations (3-parent approach using random 3 out of 6 children)

Population Size	Probability of Crossover	2-Parent Average	3-Parent Average	Winner per Pop. Size
60	0.6	0.00200770789	0.00200769037	
60	0.7	0.0020076888	0.00200769026	
60	0.8	0.00200768579	0.00200769276	2-parent
60	0.9	0.0020076863	0.00200768787	
120	0.6	0.00200767505	0.00200767566	
120	0.7	0.00200768822	0.00200766788	3-parent
120	0.8	0.00200767856	0.00200767157	
120	0.9	0.00200767644	0.00200767178	
180	0.6	0.00200766927	0.00200767186	
180	0.7	0.00200767445	0.00200766649	
180	0.8	0.00200766875	0.00200766786	
180	0.9	0.00200766956	0.00200766324	3-parent
240	0.6	0.00200767133	0.00200766659	
240	0.7	0.00200766484	0.00200766286	
240	0.8	0.00200766687	0.00200766753	
240	0.9	0.00200766555	0.00200766285	3-parent

TABLE XLIV. Traditional Crossover on *F5* With a Maximum of 100 Generations

(3-parent approach using random 3 out of 6 children)

Population Size	Probability of Crossover	2-Parent Average	3-Parent Average	Winner per Pop. Size
60	0.6	0.00200768207	0.00200770197	
60	0.7	0.0020076862	0.00200768041	
60	0.8	0.00200767754	0.00200768566	
60	0.9	0.00200767921	0.0020076736	3-parent
120	0.6	0.00200767349	0.00200766454	3-parent
120	0.7	0.00200766779	0.00200766611	
120	0.8	0.00200766758	0.00200766556	
120	0.9	0.00200767286	0.00200766749	
180	0.6	0.00200766546	0.00200766569	
180	0.7	0.00200767222	0.00200766186	3-parent
180	0.8	0.00200766374	0.00200766709	
180	0.9	0.00200766677	0.00200766511	
240	0.6	0.00200766677	0.00200766393	
240	0.7	0.00200766169	0.00200766166	
240	0.8	0.00200765964	0.00200766181	
240	0.9	0.00200766419	0.00200765931	3-parent

TABLE XLV. Traditional Crossover on *F5* With a Maximum of 150 Generations (3-parent approach using random 3 out of 6 children)

Population Size	Probability of Crossover	2-Parent Average	3-Parent Average	Winner per Pop. Size
60	0.6	0.00200767661	0.00200767763	
60	0.7	0.00200767775	0.00200767159	
60	0.8	0.00200766918	0.00200768306	2-parent
60	0.9	0.00200767462	0.00200766984	
120	0.6	0.0020076773	0.00200766646	
120	0.7	0.00200767137	0.00200766466	
120	0.8	0.00200767144	0.0020076623	3-parent
120	0.9	0.00200766913	0.00200766417	
180	0.6	0.00200766612	0.00200766171	
180	0.7	0.00200766432	0.00200766099	
180	0.8	0.0020076642	0.00200766429	
180	0.9	0.00200766357	0.00200765936	3-parent
240	0.6	0.00200766511	0.00200766088	
240	0.7	0.00200766147	0.00200766278	
240	0.8	0.00200766139	0.00200766531	
240	0.9	0.0020076591	0.00200765769	3-parent

TABLE XLVI. Traditional Crossover on *F5* With a Maximum of 200 Generations

(3-parent approach using random 3 out of 6 children)

Population Size	Probability of Crossover	2-Parent Average	3-Parent Average	Winner per Pop. Size
60	0.6	0.00200768026	0.00200767167	
60	0.7	0.00200768668	0.0020076711	
60	0.8	0.00200767214	0.00200768707	
60	0.9	0.00200767251	0.0020076701	3-parent
120	0.6	0.00200767178	0.00200766746	
120	0.7	0.00200766397	0.00200766415	
120	0.8	0.00200766552	0.00200766461	
120	0.9	0.00200766211	0.00200766365	2-parent
180	0.6	0.00200766378	0.00200765834	
180	0.7	0.00200766392	0.00200765967	
180	0.8	0.0020076611	0.00200766712	
180	0.9	0.00200765899	0.00200765814	3-parent
240	0.6	0.0020076629	0.00200765886	
240	0.7	0.00200766073	0.00200765943	
240	0.8	0.00200765805	0.00200766426	
240	0.9	0.00200766114	0.00200765647	3-parent

TABLE XLVII. Traditional Crossover on *F1* With a Maximum of 50 Generations

(3-parent approach using best 3 out of 6 children, 25 generations)

Population Size	Probability of Crossover	2-Parent Average	3-Parent Average	Winner per Pop. Size
60	0.6	0.09095	0.004995	
60	0.7	0.101505	0.004015	
60	0.8	0.0784	0.00127	
60	0.9	0.083005	0.000635	3-parent
120	0.6	0.0485	0.00104	
120	0.7	0.050485	0.00104	
120	0.8	0.03289	0.0005	
120	0.9	0.03967	0.00012	3-parent
180	0.6	0.02842	0.00117	
180	0.7	0.02916	0.001125	
180	0.8	0.02116	0.000105	3-parent
180	0.9	0.025055	0.000145	
240	0.6	0.020545	0.000735	
240	0.7	0.01673	0.00085	
240	0.8	0.01901	0.00018	
240	0.9	0.022335	0.000045	3-parent

TABLE XLVIII. Traditional Crossover on *F1* With a Maximum of 100 Generations

(3-parent approach using best 3 out of 6 children, 25 generations)

Population Size	Probability of Crossover	2-Parent Average	3-Parent Average	Winner per Pop. Size
60	0.6	0.045145	0.004995	
60	0.7	0.03248	0.004015	
60	0.8	0.01889	0.00127	
60	0.9	0.047945	0.000635	3-parent
120	0.6	0.03599	0.00104	
120	0.7	0.011815	0.00104	
120	0.8	0.01224	0.0005	
120	0.9	0.01824	0.00012	3-parent
180	0.6	0.0186	0.00117	
180	0.7	0.011185	0.001125	
180	0.8	0.011355	0.000105	3-parent
180	0.9	0.008815	0.000145	
240	0.6	0.016675	0.000735	
240	0.7	0.00562	0.00085	
240	0.8	0.00593	0.00018	
240	0.9	0.00719	0.000045	3-parent

TABLE XLIX. Traditional Crossover on *F1* With a Maximum of 150 Generations

(3-parent approach using best 3 out of 6 children, 25 generations)

Population Size	Probability of Crossover	2-Parent Average	3-Parent Average	Winner per Pop. Size
60	0.6	0.03494	0.004995	
60	0.7	0.016225	0.004015	
60	0.8	0.015935	0.00127	
60	0.9	0.07179	0.000635	3-parent
120	0.6	0.020935	0.00104	
120	0.7	0.010565	0.00104	
120	0.8	0.00863	0.0005	
120	0.9	0.020415	0.00012	3-parent
180	0.6	0.01292	0.00117	
180	0.7	0.00757	0.001125	
180	0.8	0.007345	0.000105	3-parent
180	0.9	0.00834	0.000145	
240	0.6	0.00669	0.000735	
240	0.7	0.00434	0.00085	
240	0.8	0.00342	0.00018	
240	0.9	0.005545	0.000045	3-parent

TABLE L. Traditional Crossover on *F1* With a Maximum of 200 Generations

(3-parent approach using best 3 out of 6 children, 25 generations)

Population Size	Probability of Crossover	2-Parent Average	3-Parent Average	Winner per Pop. Size
60	0.6	0.02378	0.004995	
60	0.7	0.010925	0.004015	
60	0.8	0.00661	0.00127	
60	0.9	0.02964	0.000635	3-parent
120	0.6	0.01722	0.00104	
120	0.7	0.00694	0.00104	
120	0.8	0.00663	0.0005	
120	0.9	0.012235	0.00012	3-parent
180	0.6	0.018	0.00117	
180	0.7	0.006645	0.001125	
180	0.8	0.005935	0.000105	3-parent
180	0.9	0.00489	0.000145	
240	0.6	0.00064	0.000735	
240	0.7	0.004055	0.00085	
240	0.8	0.00333	0.00018	
240	0.9	0.00318	0.000045	3-parent

TABLE LI. Traditional Crossover on *F2* With a Maximum of 50 Generations (3-parent approach using best 3 out of 6 children, 25 generations)

Population Size	Probability of Crossover	2-Parent Average	3-Parent Average	Winner per Pop. Size
60	0.6	0.043925	0.046105	
60	0.7	0.03957	0.024118	3-parent
60	0.8	0.035831	0.030675	
60	0.9	0.048861	0.035952	
120	0.6	0.036901	0.012051	
120	0.7	0.020055	0.016715	
120	0.8	0.009347	0.006999	3-parent
120	0.9	0.0152	0.0185	
180	0.6	0.016152	0.004394	3-parent
180	0.7	0.011937	0.005232	
180	0.8	0.008612	0.007614	
180	0.9	0.012575	0.005678	
240	0.6	0.008243	0.005688	
240	0.7	0.005415	0.007258	
240	0.8	0.008889	0.003221	3-parent
240	0.9	0.013365	0.003437	

TABLE LII. Traditional Crossover on *F2* With a Maximum of 100 Generations
(3-parent approach using best 3 out of 6 children, 25 generations)

Population Size	Probability of Crossover	2-Parent Average	3-Parent Average	Winner per Pop. Size
60	0.6	0.020596	0.046105	
60	0.7	0.027332	0.024118	
60	0.8	0.018699	0.030675	2-parent
60	0.9	0.032807	0.035952	
120	0.6	0.00973	0.012051	
120	0.7	0.013187	0.016715	
120	0.8	0.005214	0.006999	2-parent
120	0.9	0.011156	0.0185	
180	0.6	0.013082	0.004394	3-parent
180	0.7	0.007625	0.005232	
180	0.8	0.004441	0.007614	
180	0.9	0.006742	0.005678	
240	0.6	0.005715	0.005688	
240	0.7	0.003051	0.007258	2-parent
240	0.8	0.003141	0.003221	
240	0.9	0.004655	0.003437	

TABLE LIII. Traditional Crossover on *F2* With a Maximum of 150 Generations (3-parent approach using best 3 out of 6 children, 25 generations)

Population Size	Probability of Crossover	2-Parent Average	3-Parent Average	Winner per Pop. Size
60	0.6	0.018237	0.046105	
60	0.7	0.009509	0.024118	2-parent
60	0.8	0.011167	0.030675	
60	0.9	0.019174	0.035952	
120	0.6	0.0127	0.012051	
120	0.7	0.00755	0.016715	
120	0.8	0.003679	0.006999	2-parent
120	0.9	0.005656	0.0185	
180	0.6	0.004709	0.004394	
180	0.7	0.003797	0.005232	
180	0.8	0.003123	0.007614	2-parent
180	0.9	0.003994	0.005678	
240	0.6	0.002761	0.005688	
240	0.7	0.002202	0.007258	2-parent
240	0.8	0.003052	0.003221	
240	0.9	0.002302	0.003437	

TABLE LIV. Traditional Crossover on *F2* With a Maximum of 200 Generations

(3-parent approach using best 3 out of 6 children, 25 generations)

Population Size	Probability of Crossover	2-Parent Average	3-Parent Average	Winner per Pop. Size
60	0.6	0.021105	0.046105	
60	0.7	0.017838	0.024118	
60	0.8	0.010575	0.030675	2-parent
60	0.9	0.018035	0.035952	
120	0.6	0.009948	0.012051	
120	0.7	0.003948	0.016715	
120	0.8	0.003508	0.006999	
120	0.9	0.003038	0.0185	2-parent
180	0.6	0.004197	0.004394	
180	0.7	0.00334	0.005232	
180	0.8	0.002857	0.007614	
180	0.9	0.002693	0.005678	2-parent
240	0.6	0.003387	0.005688	
240	0.7	0.002148	0.007258	
240	0.8	0.001449	0.003221	2-parent
240	0.9	0.00216	0.003437	

TABLE LV. Traditional Crossover on *F3* With a Maximum of 50 Generations

(3-parent approach using best 3 out of 6 children, 25 generations)

Population Size	Probability of Crossover	2-Parent Average	3-Parent Average	Winner per Pop. Size
60	0.6	7.65	2.45	
60	0.7	5.95	1.35	
60	0.8	6.6	2.25	
60	0.9	5.5	0.9	3-parent
120	0.6	5.4	0.85	
120	0.7	5.05	0.35	
120	0.8	3.65	0.45	
120	0.9	2.65	0.15	3-parent
180	0.6	3.05	0.5	
180	0.7	3.3	0.2	
180	0.8	3.2	0.25	
180	0.9	2.7	0.05	3-parent
240	0.6	2.7	0.3	
240	0.7	3.2	0.05	3-parent
240	0.8	2.25	0.1	
240	0.9	1.5	0.05	3-parent

TABLE LVI. Traditional Crossover on *F3* With a Maximum of 100 Generations

(3-parent approach using best 3 out of 6 children, 25 generations)

Population Size	Probability of Crossover	2-Parent Average	3-Parent Average	Winner per Pop. Size
60	0.6	6.8	2.45	
60	0.7	6.95	1.35	
60	0.8	7.6	2.25	
60	0.9	5.25	0.9	3-parent
120	0.6	4.75	0.85	
120	0.7	4.55	0.35	
120	0.8	3.05	0.45	
120	0.9	3.75	0.15	3-parent
180	0.6	3.0	0.5	
180	0.7	3.6	0.2	
180	0.8	2.65	0.25	
180	0.9	2.85	0.05	3-parent
240	0.6	3.15	0.3	
240	0.7	3.05	0.05	3-parent
240	0.8	1.5	0.1	
240	0.9	1.85	0.05	3-parent

TABLE LVII. Traditional Crossover on *F3* With a Maximum of 150 Generations (3-parent approach using best 3 out of 6 children, 25 generations)

Population Size	Probability of Crossover	2-Parent Average	3-Parent Average	Winner per Pop. Size
60	0.6	7.6	2.45	
60	0.7	6.35	1.35	
60	0.8	6.55	2.25	
60	0.9	6.5	0.9	3-parent
120	0.6	5.5	0.85	
120	0.7	4.35	0.35	
120	0.8	3.95	0.45	
120	0.9	3.1	0.15	3-parent
180	0.6	2.05	0.5	
180	0.7	1.0	0.2	
180	0.8	1.2	0.25	
180	0.9	1.1	0.05	3-parent
240	0.6	3.2	0.3	
240	0.7	2.55	0.05	3-parent
240	0.8	1.9	0.1	
240	0.9	1.9	0.05	3-parent

TABLE LVIII. Traditional Crossover on *F3* With a Maximum of 200 Generations

(3-parent approach using best 3 out of 6 children, 25 generations)

Population Size	Probability of Crossover	2-Parent Average	3-Parent Average	Winner per Pop. Size
60	0.6	6.75	2.45	
60	0.7	5.8	1.35	
60	0.8	6.5	2.25	
60	0.9	6.1	0.9	3-parent
120	0.6	5.45	0.85	
120	0.7	3.6	0.35	
120	0.8	3.9	0.45	
120	0.9	3.0	0.15	3-parent
180	0.6	3.05	0.5	
180	0.7	3.3	0.2	
180	0.8	3.05	0.25	
180	0.9	2.0	0.05	3-parent
240	0.6	3.75	0.3	
240	0.7	2.45	0.05	3-parent
240	0.8	2.55	0.1	
240	0.9	1.65	0.05	3-parent

TABLE LIX. Traditional Crossover on *F4* With a Maximum of 50 Generations

(3-parent approach using best 3 out of 6 children, 25 generations)

Population Size	Probability of Crossover	2-Parent Average	3-Parent Average	Winner per Pop. Size
60	0.6	52.32811	17.33576	
60	0.7	50.2521	15.26377	
60	0.8	45.99219	10.21506	
60	0.9	52.07662	8.211661	3-parent
120	0.6	44.01746	11.79268	
120	0.7	46.01452	11.14837	
120	0.8	40.77805	6.328627	
120	0.9	44.93026	4.657941	3-parent
180	0.6	44.84551	12.92878	
180	0.7	43.20634	7.082359	
180	0.8	44.52702	6.258305	
180	0.9	43.57111	3.84107	3-parent
240	0.6	39.72294	10.87544	
240	0.7	43.41834	7.084543	
240	0.8	41.89307	5.01345	
240	0.9	42.98437	3.474466	3-parent

TABLE LX. Traditional Crossover on *F4* With a Maximum of 100 Generations

(3-parent approach using best 3 out of 6 children, 25 generations)

Population Size	Probability of Crossover	2-Parent Average	3-Parent Average	Winner per Pop. Size
60	0.6	32.21346	17.33576	
60	0.7	16.11699	15.26377	
60	0.8	19.38805	10.21506	
60	0.9	32.10905	8.211661	3-parent
120	0.6	13.40055	11.79268	
120	0.7	12.5976	11.14837	
120	0.8	10.67787	6.328627	
120	0.9	9.077796	4.657941	3-parent
180	0.6	13.7977	12.92878	
180	0.7	14.3492	7.082359	
180	0.8	13.53767	6.258305	
180	0.9	12.6264	3.84107	3-parent
240	0.6	16.89635	10.87544	
240	0.7	13.84845	7.084543	
240	0.8	13.91625	5.01345	
240	0.9	14.74651	3.474466	3-parent

TABLE LXI. Traditional Crossover on *F4* With a Maximum of 150 Generations

(3-parent approach using best 3 out of 6 children, 25 generations)

Population Size	Probability of Crossover	2-Parent Average	3-Parent Average	Winner per Pop. Size
60	0.6	26.72981	17.33576	
60	0.7	34.19218	15.26377	
60	0.8	15.23151	10.21506	
60	0.9	21.90235	8.211661	3-parent
120	0.6	7.607474	11.79268	
120	0.7	5.559006	11.14837	
120	0.8	6.025749	6.328627	
120	0.9	5.672263	4.657941	3-parent
180	0.6	5.126395	12.92878	
180	0.7	5.499776	7.082359	
180	0.8	6.582014	6.258305	
180	0.9	6.012601	3.84107	3-parent
240	0.6	6.005873	10.87544	
240	0.7	6.662924	7.084543	
240	0.8	6.308006	5.01345	
240	0.9	6.654461	3.474466	3-parent

TABLE LXII. Traditional Crossover on *F4* With a Maximum of 200 Generations (3-parent approach using best 3 out of 6 children, 25 generations)

Population Size	Probability of Crossover	2-Parent Average	3-Parent Average	Winner per Pop. Size
60	0.6	24.83604	17.33576	
60	0.7	25.71368	15.26377	
60	0.8	19.29439	10.21506	
60	0.9	15.29822	8.211661	3-parent
120	0.6	3.419861	11.79268	2-parent
120	0.7	6.471318	11.14837	
120	0.8	5.009745	6.328627	
120	0.9	3.706555	4.657941	
180	0.6	4.854252	12.92878	
180	0.7	4.201519	7.082359	
180	0.8	5.062804	6.258305	
180	0.9	4.755725	3.84107	3-parent
240	0.6	4.508327	10.87544	
240	0.7	4.951602	7.084543	
240	0.8	5.353422	5.01345	
240	0.9	4.745058	3.474466	3-parent

TABLE LXIII. Traditional Crossover on *F5* With a Maximum of 50 Generations (3-parent approach using best 3 out of 6 children, 25 generations)

Population Size	Probability of Crossover	2-Parent Average	3-Parent Average	Winner per Pop. Size
60	0.6	0.00200770789	0.00200765897	
60	0.7	0.0020076888	0.00200765707	
60	0.8	0.00200768579	0.00200765589	
60	0.9	0.0020076863	0.00200765566	3-parent
120	0.6	0.00200767505	0.00200765588	
120	0.7	0.00200768822	0.00200765485	
120	0.8	0.00200767856	0.00200765573	
120	0.9	0.00200767644	0.00200765464	3-parent
180	0.6	0.00200766927	0.00200765485	
180	0.7	0.00200767445	0.0020076548	
180	0.8	0.00200766875	0.00200765472	
180	0.9	0.00200766956	0.00200765462	3-parent
240	0.6	0.00200767133	0.00200765477	
240	0.7	0.00200766484	0.0020076546	3-parent
240	0.8	0.00200766687	0.0020076546	3-parent
240	0.9	0.00200766555	0.00200765462	

TABLE LXIV. Traditional Crossover on *F5* With a Maximum of 100 Generations

(3-parent approach using best 3 out of 6 children, 25 generations)

Population Size	Probability of Crossover	2-Parent Average	3-Parent Average	Winner per Pop. Size
60	0.6	0.00200768207	0.00200765897	
60	0.7	0.0020076862	0.00200765707	
60	0.8	0.00200767754	0.00200765589	
60	0.9	0.00200767921	0.00200765566	3-parent
120	0.6	0.00200767349	0.00200765588	
120	0.7	0.00200766779	0.00200765485	
120	0.8	0.00200766758	0.00200765573	
120	0.9	0.00200767286	0.00200765464	3-parent
180	0.6	0.00200766546	0.00200765485	
180	0.7	0.00200767222	0.0020076548	
180	0.8	0.00200766374	0.00200765472	
180	0.9	0.00200766677	0.00200765462	3-parent
240	0.6	0.00200766677	0.00200765477	
240	0.7	0.00200766169	0.0020076546	3-parent
240	0.8	0.00200765964	0.0020076546	3-parent
240	0.9	0.00200766419	0.00200765462	

TABLE LXV. Traditional Crossover on *F5* With a Maximum of 150 Generations
(3-parent approach using best 3 out of 6 children, 25 generations)

Population Size	Probability of Crossover	2-Parent Average	3-Parent Average	Winner per Pop. Size
60	0.6	0.00200767661	0.00200765897	
60	0.7	0.00200767775	0.00200765707	
60	0.8	0.00200766918	0.00200765589	
60	0.9	0.00200767462	0.00200765566	3-parent
120	0.6	0.0020076773	0.00200765588	
120	0.7	0.00200767137	0.00200765485	
120	0.8	0.00200767144	0.00200765573	
120	0.9	0.00200766913	0.00200765464	3-parent
180	0.6	0.00200766612	0.00200765485	
180	0.7	0.00200766432	0.0020076548	
180	0.8	0.0020076642	0.00200765472	
180	0.9	0.00200766357	0.00200765462	3-parent
240	0.6	0.00200766511	0.00200765477	
240	0.7	0.00200766147	0.0020076546	3-parent
240	0.8	0.00200766139	0.0020076546	3-parent
240	0.9	0.0020076591	0.00200765462	

TABLE LXVI. Traditional Crossover on *F5* With a Maximum of 200 Generations

(3-parent approach using best 3 out of 6 children, 25 generations)

Population Size	Probability of Crossover	2-Parent Average	3-Parent Average	Winner per Pop. Size
60	0.6	0.00200768026	0.00200765897	
60	0.7	0.00200768668	0.00200765707	
60	0.8	0.00200767214	0.00200765589	
60	0.9	0.00200767251	0.00200765566	3-parent
120	0.6	0.00200767178	0.00200765588	
120	0.7	0.00200766397	0.00200765485	
120	0.8	0.00200766552	0.00200765573	
120	0.9	0.00200766211	0.00200765464	3-parent
180	0.6	0.00200766378	0.00200765485	
180	0.7	0.00200766392	0.0020076548	
180	0.8	0.0020076611	0.00200765472	
180	0.9	0.00200765899	0.00200765462	3-parent
240	0.6	0.0020076629	0.00200765477	
240	0.7	0.00200766073	0.0020076546	3-parent
240	0.8	0.00200765805	0.0020076546	3-parent
240	0.9	0.00200766114	0.00200765462	

APPENDIX D

Genetic Algorithm Program Listings

```
{ Pascal program used to obtain the results for the Ph.D. dissertation    }
{ GENETIC ALGORITHMS WITH 3-PARENT CROSSOVER                              }
{ by L. Vincent Edmondson.                                                }
{                                                                         }
{ COPYRIGHT 1993 by Lawrence Vincent Edmondson.                           }
{ All Rights Reserved.                                                    }
{                                                                         }
{ This program is an implementation of two new families of genetic        }
{ algorithms. The first new family uses a 3-parent uniform crossover      }
{ operator during the reproduction phase. This is fully described in      }
{ Chapter I of the dissertation cited above. In addition to this          }
{ new crossover operator, the program also implements the standard        }
{ 2-parent uniform crossover operator (for comparison purposes).          }
{                                                                         }
{ The other new family uses 3-parent traditional crossover operators      }
{ during the reproduction process. These are fully described in           }
{ Chapter II of the dissertation cited above. In addition to these        }
{ new crossover operators, the program also implements the standard       }
{ 2-parent crossover operator (for comparison purposes).                  }
{                                                                         }
{ The program requires an input file named "genalg.in" to be present     }
{ in the current directory. The structure of this file is as follows:     }
{                                                                         }
{ line 1:  maximum number of generations                                  }
{ line 2:  function number,  number of parents                            }
{ line 3:  maximum population size, crossover type                        }
{ line 4:  initial probability of crossover                               }
{ line 5:  probability of mutation                                        }
{                                                                         }
{ Input file restrictions/considerations:                                 }
{                                                                         }
{ The maximum number of generations should be an integer value (there is  }
{ an upper limit of 500 (see "totals" array below), although the nature   }
{ of a GA is such that a practical upper limit is <= 200 for the          }
{ functions used here.                                                    }
{                                                                         }
{ The function number should be an integer in the range from 1 to 5       }
{ (inclusive). This number represents the function number from the        }
{ De Jong test suite. The number of parents should be either 2 or 3.      }
{                                                                         }
{ The maximum population size should be an integer value <= 252. For      }
{ comparison purposes, it was always set to a multiple of 6 so that       }
{ the 2-parent and 3-parent approaches would have the same population     }
{ size. The crossover type should be one of the following alphabetic      }
{ characters:                                                             }
{     u : uniform crossover                                               }
```

```
{      t :  traditional crossover (for 3-parent, this gives a random        }
{            3 of 6 children                                                }
{      b :  traditional crossover (this is only valid for 3-parent          }
{            crossover and will give the best 3 of 6 children)              }
{      s :  traditional crossover (this is only valid for 3-parent          }
{            crossover and will give all 6 children resulting from the      }
{            3-parent, 2-point reproduction process ... this was not        }
{            included in the dissertation results)                          }
{                                                                           }
{  The probability of crossover should be a real number x such that         }
{  0 <= x <= 1.  If the probability of crossover is >= 1, then crossover    }
{  will always be performed.                                                }
{                                                                           }
{  The probability of mutation should be a real number x such that          }
{  0 <= x <= 1.  Typically, this value is very small (i.e., 0.01)           }
{                                                                           }
{  Output is always directed to the file named "ga.out".                    }
{                                                                           }

program genetic_alg (input, output, infile, outfile);

{  The const maxpopulation is limited only by the particular hardware used  }
{  to run the program.  Original development was done using Borland's       }
{  Turbo Pascal (version 5.5) for the IBM PC.  The Professional Pascal      }
{  compiler from MetaWare was used for the eventual program runs on IBM     }
{  machines.  The maxstring value is 240 because that is the longest        }
{  string required for the 5 functions in the De Jong test suite.           }
{  The number_of_trials value was used to control the number of executions  }
{  for each set of parameters.  The results were all averaged over the      }
{  entire number_of_trials.                                                 }


const maxpopulation    = 252;
      maxstring        = 240;
      number_of_trials = 20;

{  The following type and var sections use variables whose names are       }
{  descriptive of their purpose.  It should be noted that pop_ptr was      }
{  used to speed up the replacement of the population from generation      }
{  t to generation t+1.  The best_mins and best_gens types were used       }
{  to specify the arrays which kept track of the minimum function value    }
{  for a given trial and the generation in which this minimum was found.   }
{  The function5array was used to hold the 2-dimensional array used with   }
{  function 5 from the De Jong test suite.                                 }
```

```
type
     bit_string = array [1..maxstring] of boolean;
     member = record
              bits : bit_string;
              real_fitness : real;
              fitness : real;
              end;
     popu = record
            pop : array [1..maxpopulation] of member;
            end;
     pop_ptr = ^popu;
     best_mins = array [1..number_of_trials] of real;
     best_gens = array [1..number_of_trials] of integer;
     stringone = char;
     function5array = array [1..2,1..25] of real;

var  infile, outfile: text;
     p, q : pop_ptr;
     number_of_genes, number_of_members, number_of_bits: integer;
     global_best_gen, number_of_parents : integer;
     function_number, gen, maxgen: integer;
     j, jj, k, m : integer;
     mask : array [1..3, 1..240] of integer;
     seed, pcross, pmutation, sumfitness: real;
     avg, denominator, min: real;
     best_value, global_best_value: real;
     best_bits: bit_string;
     f_max, f_max_addition, max : real;
     online_sum, online_average, offline_sum, offline_average : real;
     totals : array [1..2, 0..500] of real;
     output_filename : packed array [1..12] of char;
     blank_space, cross_type, f_string, p_string : stringone;
     best_of_trial : best_mins;
     best_of_gen : best_gens;
     a: function5array;

{  The following function is used to generate a uniformly-distributed          }
{  random number between 0 and 1.  It is included in the program to            }
{  ensure replicability of the experiments performed for this research.        }
{  It is based on L'Ecuyer's Minimum Standard, as reported in the article       }
{  "Efficient and Portable Combined Random Number Generators."  This            }
{  article can be found in COMMUNICATIONS OF THE ACM, Volume 31,                }
{  No. 6, pages 742-749, 774.                                                   }
```

```
function random (var ix: real):real;
 function realmod (x,y : real) : real;
  begin
   realmod := (x - y * trunc(x/y));
  end;

begin
ix := ix * 40692.0;
ix := realmod (ix, 2.147483399e9);
random := ix * 4.656613413e-10;
end;

{  The following function returns a value of true when the random number     }
{  generated is <= than the argument.  It is primarily used to determine     }
{  if crossover will be invoked.                                              }

function flip (probability: real): boolean;
begin
   flip := (random (seed) <= probability);
end;

{  The following function decodes the bit string that is sent as a           }
{  parameter and then evaluates the function (using the value of the         }
{  variable "function_number".  The functions are from the De Jong test      }
{  suite.                                                                     }
{                                                                             }
{  This program uses De Jong's original encoding scheme (and not Gray        }
{  coding).                                                                   }

function f (var bits: bit_string; number_of_bits : integer): real;
const  max_number_of_genes = 30;
type   g = array [1..max_number_of_genes] of real;
var    genes : g;
       gene_length, i, j, k, integer_gene: integer;
       noise, sum, powerof2, sum1, diff, prod : real;
begin
gene_length := number_of_bits DIV number_of_genes;
for j := 1 to number_of_genes do
    begin
    genes[j] := 0.0;
    powerof2 := 1.0;
    for k := ((j-1)*gene_length + 2) to (j*gene_length) do
        begin
        if bits[k] then genes[j] := genes[j] + powerof2;
        powerof2 := powerof2 * 2.0;
        end;
```

```
if not bits[(j-1)*gene_length + 1] then genes[j] := genes[j]*(-1.0);
genes[j] := genes[j] / denominator;
end;
case function_number of
1 : begin
     sum := 0.0;
     for j := 1 to number_of_genes do
        sum := sum + sqr (genes[j]);
     f := sum;
    end;
2 : begin
     f := 100.0 * sqr(sqr(genes[1]) - genes[2]) + sqr (1.0 - genes[1]);
    end;
3 : begin
     sum := 0.0;
     for j := 1 to 5 do
      begin
       integer_gene := trunc (genes[j]);
       if integer_gene > genes[j] then integer_gene := integer_gene - 1;
       sum := sum + integer_gene;
      end;
     f := sum + 30.0;
    end;
4 : begin
     sum := 0.0;
     for j := 1 to number_of_genes do
        sum := sum + j*(sqr(sqr(genes[j])));
     noise := 0.0;
     for j := 1 to 12 do
       noise := noise + random (seed);
     noise := noise - 6.0;
     f := sum + noise;
   end;
5 : begin
    sum := 0.002;
    for j := 1 to 25 do
     begin
     sum1 := j;
     for i := 1 to 2 do
      begin
       diff := genes[i] - a[i,j];
       prod := 1.0;
       for k := 1 to 6 do
         prod := prod * diff;
       sum1 := sum1 + prod;
      end;
```

```
        sum := sum + 1.0/sum1;
        end;
        f := sum;
        end;
    end;
end;

{  The following procedure finds the function value for each member of the        }
{  population.  The field "real_fitness" contains the actual f(x) value,          }
{  while the field "fitness" contains a scaled version of f(x).  This              }
{  fitness scaling is necessary so that the problems associated with               }
{  extraordinary individuals (i.e., dominating the population) can be              }
{  avoided.                                                                        }

procedure evaluate (var p: pop_ptr; number_of_members,
                    number_of_bits: integer);
var j : integer;
begin
for j := 1 to number_of_members do
 begin
    p^.pop[j].real_fitness := f(p^.pop[j].bits, number_of_bits);
    p^.pop[j].fitness := f_max - p^.pop[j].real_fitness;
 end;
end;

{  The following procedure initializes the population.  Each bit position          }
{  is given a value of either false or true (0 or 1), each occuring with            }
{  equal probability.                                                               }

procedure initialize (var p: pop_ptr; number_of_members,
                      number_of_bits: integer);
var j, k : integer;
begin
for j := 1 to number_of_members do
    for k := 1 to number_of_bits do
        if random (seed) < 0.5 then
           p^.pop[j].bits[k] := false
         else
           p^.pop[j].bits[k] := true;
    evaluate (p, number_of_members, number_of_bits);
end;

{  The following function selects an individual for reproduction.  It is           }
{  based on the idea of a biased roulette wheel.                                    }
```

```
function select (var p: pop_ptr; number_of_members: integer;
               sumfitness: real): integer;
var rand, partsum: real;
    j : integer;
begin
 partsum := 0.0;
 j := 0;
 rand := random (seed) *sumfitness;
 repeat
   j := j + 1;
   partsum := partsum + p^.pop[j].fitness;
 until (partsum >= rand) or (j = number_of_members);
 select := j;
end;

{  The following function mutates a bit (changes it from false to true or          }
{  vice versa) if a uniformly-distributed random number between 0 and 1 is          }
{  less than the probability of mutation (which is typically very small).          }

function mutation (bit : boolean; pmutation: real): boolean;
var mutate : boolean;
begin
mutate := flip(pmutation);
if mutate then
   mutation := not bit
 else
   mutation := bit;
end;

{  The following procedure performs 2-parent traditional crossover.  Bit positions   }
{  from jcross2 to number_of_bits are already stored in the correct positions in     }
{  parents.                                                                            }

procedure change2 (var parent1, parent2, child1, child2: bit_string;
                   number_of_bits, jcross1, jcross2: integer);
var j : integer;
 begin
 for j := 1 to jcross1 do
  begin
   child1[j] := mutation(parent1[j], pmutation);
   child2[j] := mutation(parent2[j], pmutation);
  end;
 for j:= (jcross1+1) to jcross2 do
  begin
   child1[j] := mutation(parent2[j], pmutation);
   child2[j] := mutation(parent1[j], pmutation);
```

```
  end;
 for j := (jcross2+1) to number_of_bits do
  begin
   child1[j] := mutation(parent1[j], pmutation);
   child2[j] := mutation(parent2[j], pmutation);
  end;
 end;

{  The following procedure performs 3-parent traditional crossover.          }
{  The value of v determines which of the 6 children are generated.          }

procedure change3 (var parent1, parent2, parent3, child: bit_string;
                   number_of_bits, jcross1, jcross2, v : integer);
var j : integer;
 begin
case v of
0: begin
 for j := 1 to jcross1 do
  child[j] := mutation(parent1[j], pmutation);
 for j := (jcross1+1) to jcross2 do
  child[j] := mutation(parent2[j], pmutation);
 for j := (jcross2+1) to number_of_bits do
  child[j] := mutation(parent3[j], pmutation);
 end;
1: begin
 for j := 1 to jcross1 do
  child[j] := mutation(parent1[j], pmutation);
 for j := (jcross1+1) to jcross2 do
  child[j] := mutation(parent3[j], pmutation);
 for j := (jcross2+1) to number_of_bits do
  child[j] := mutation(parent2[j], pmutation);
 end;
2: begin
 for j := 1 to jcross1 do
  child[j] := mutation(parent2[j], pmutation);
 for j := (jcross1+1) to jcross2 do
  child[j] := mutation(parent1[j], pmutation);
 for j := (jcross2+1) to number_of_bits do
  child[j] := mutation(parent3[j], pmutation);
 end;
3: begin
 for j := 1 to jcross1 do
  child[j] := mutation(parent2[j], pmutation);
 for j := (jcross1+1) to jcross2 do
  child[j] := mutation(parent3[j], pmutation);
 for j := (jcross2+1) to number_of_bits do
```

```
  child[j] := mutation(parent1[j], pmutation);
 end;
4: begin
 for j := 1 to jcross1 do
  child[j] := mutation(parent3[j], pmutation);
 for j := (jcross1+1) to jcross2 do
  child[j] := mutation(parent1[j], pmutation);
 for j := (jcross2+1) to number_of_bits do
  child[j] := mutation(parent2[j], pmutation);
 end;
5: begin
 for j := 1 to jcross1 do
  child[j] := mutation(parent3[j], pmutation);
 for j := (jcross1+1) to jcross2 do
  child[j] := mutation(parent2[j], pmutation);
 for j := (jcross2+1) to number_of_bits do
  child[j] := mutation(parent1[j], pmutation);
 end;
 end;
 end;

{  The following procedure starts the process of performing 2-parent          }
{  crossover (either uniform or traditional, depending on "cross_type".      }

procedure crossover2 (var parent1, parent2, child1, child2: bit_string;
                      var number_of_bits: integer;
                      var pcross, pmutation: real; cross_type: stringone);
var jcross1, jcross2, j, k : integer;
   parents : array[0..2] of bit_string;
begin

 if flip(pcross) then
   if cross_type = 'u' then
     begin
      parents[0] := parent1;
      parents[1] := parent2;
      for j := 1 to number_of_bits do
         mask[1,j]  := trunc (2 * random(seed));
      for j := 1 to number_of_bits do
       begin
         mask[2,j] := (mask[1,j] + 1) mod 2;
       end;
      for j := 1 to number_of_bits do
       begin
         child1[j] := mutation (parents[mask[1,j],j], pmutation);
         child2[j] := mutation (parents[mask[2,j],j], pmutation);
```

```
        end;
      end
     else
      begin
      jcross1 := trunc ((number_of_bits - 1) * random(seed)) + 1;
      jcross2 := trunc ((number_of_bits - 1) * random(seed)) + 1;
      if jcross1 > jcross2 then begin
        j := jcross1;
        jcross1 := jcross2;
        jcross2 := jcross1;
       end;
      change2 (parent1, parent2, child1, child2, number_of_bits, jcross1,
               jcross2);
    end
   else
    for j := 1 to number_of_bits do begin
     child1[j] := mutation(parent1[j], pmutation);
     child2[j] := mutation(parent2[j], pmutation);
    end;
end;

{  The following procedure starts the process of performing 3-parent         }
{  crossover (either uniform or some form of traditional, depending on       }
{  "cross_type".                                                             }

procedure crossover3 (var parent1, parent2, parent3,
                          child1, child2, child3: bit_string;
                      var number_of_bits: integer;
                      var pcross, pmutation: real; cross_type: stringone);
var jcross1, jcross2, j, k, v, count, count2 : integer;
    parents : array[0..2] of bit_string;
    s1 : set of 0..5;
    c : array [0..5] of bit_string;
    func : array [0..5] of real;
    tempc : bit_string;
    tempf : real;
begin

 if flip(pcross) then
   if cross_type = 'u' then
     begin
      parents[0] := parent1;
      parents[1] := parent2;
      parents[2] := parent3;
      for j := 1 to number_of_bits do
         mask[1,j]  := trunc (2 * random(seed));
```

```
    for j := 1 to number_of_bits do
     begin
       mask[2,j] := (mask[1,j] + 1) mod 3;
       mask[3,j] := (mask[1,j] + 2) mod 3;
     end;
    for j := 1 to number_of_bits do
     begin
       child1[j] := mutation (parents[mask[1,j],j], pmutation);
       child2[j] := mutation (parents[mask[2,j],j], pmutation);
       child3[j] := mutation (parents[mask[3,j],j], pmutation);
     end;
   end
  else if cross_type = 't' then
   begin
   jcross1 := trunc ((number_of_bits - 1) * random (seed)) + 1;
   jcross2 := trunc ((number_of_bits - 1) * random (seed)) + 1;
   if jcross1 > jcross2 then begin
     j := jcross1;
     jcross1 := jcross2;
     jcross2 := jcross1;
    end;
   s1 := [];
   v := trunc (6 * random(seed));
   s1 := s1 + [v];
   change3 (parent1, parent2, parent3, child1, number_of_bits, jcross1,
           jcross2, v);
   v := trunc (6 * random(seed));
   while v in s1 do
     v := trunc (6 * random(seed));
   s1 := s1 + [v];
   change3 (parent1, parent2, parent3, child2, number_of_bits, jcross1,
           jcross2, v);
   v :=  trunc (6 * random(seed));
   while v in s1 do
     v := trunc (6 * random(seed));
   s1 := s1 + [v];
   change3 (parent1, parent2, parent3, child3, number_of_bits, jcross1,
           jcross2, v);
 end
 else  { cross_type must be 'b' ==> take best 3 of six children }
  begin
   jcross1 := trunc ((number_of_bits - 1) * random (seed)) + 1;
   jcross2 := trunc ((number_of_bits - 1) * random (seed)) + 1;
   if jcross1 > jcross2 then begin
     j := jcross1;
     jcross1 := jcross2;
```

```
    jcross2 := jcross1;
   end;
   for count := 0 to 5 do
     begin
       change3 (parent1, parent2, parent3, c[count], number_of_bits,
              jcross1, jcross2, count);
       func[count] := f(c[count], number_of_bits);
     end;
  { choose the three best children... Bubblesort is used here }
   for count := 0 to 4 do
      for count2 := 0 to (5-count) do
        if func[count2] > func[count2+1] then
          begin
            tempf := func[count2];
            func[count2] := func[count2+1];
            func[count2+1] := tempf;
            tempc := c[count2];
            c[count2] := c[count2+1];
            c[count2+1] := tempc;
           end;
     child1 := c[0];
     child2 := c[1];
     child3 := c[2];
  end

 else
  for j := 1 to number_of_bits do begin
   child1[j] := mutation(parent1[j], pmutation);
   child2[j] := mutation(parent2[j], pmutation);
   child3[j] := mutation(parent3[j], pmutation);
  end;
end;

{  The following procedure starts the process of performing 3-parent          }
{  crossover using the traditional approach.  It generates all 6 children.    }
{  This crossover operator was not included in the final results presented    }
{  in the dissertation.                                                       }

procedure crossover6 (var parent1, parent2, parent3,
              child1, child2, child3, child4, child5, child6: bit_string;
              var number_of_bits: integer;
              var pcross, pmutation: real; cross_type: stringone);
var jcross1, jcross2, j, k, v : integer;
   parents : array[0..2] of bit_string;
begin
```

```
 if flip(pcross) then
    begin
    jcross1 := trunc ((number_of_bits - 1) * random (seed)) + 1;
    jcross2 := trunc ((number_of_bits - 1) * random (seed)) + 1;
    if jcross1 > jcross2 then begin
      j := jcross1;
      jcross1 := jcross2;
      jcross2 := jcross1;
     end;
    change3 (parent1, parent2, parent3, child1, number_of_bits, jcross1,
           jcross2, 0);
    change3 (parent1, parent2, parent3, child2, number_of_bits, jcross1,
           jcross2, 1);
    change3 (parent1, parent2, parent3, child3, number_of_bits, jcross1,
           jcross2, 2);
    change3 (parent1, parent2, parent3, child4, number_of_bits, jcross1,
           jcross2, 3);
    change3 (parent1, parent2, parent3, child5, number_of_bits, jcross1,
           jcross2, 4);
    change3 (parent1, parent2, parent3, child6, number_of_bits, jcross1,
           jcross2, 5);
  end
 else

  for j := 1 to number_of_bits do begin
   child1[j] := mutation(parent1[j], pmutation);
   child2[j] := mutation(parent2[j], pmutation);
   child3[j] := mutation(parent3[j], pmutation);
   child4[j] := mutation(parent1[j], pmutation);
   child5[j] := mutation(parent2[j], pmutation);
   child6[j] := mutation(parent3[j], pmutation);
  end;
end;

{  This procedure creates a new generation from the old generation.          }
{  This research used the population replacement strategy of generational    }
{  replacement.                                                              }

procedure generation (var p: pop_ptr; number_of_parents,
                   number_of_members, number_of_bits: integer;
                   pcross, pmutation: real; var sumfitness: real;
                   cross_type: stringone);
var j, mate1, mate2, mate3, jcross1, jcross2: integer;
   temp_ptr : pop_ptr;
begin
case number_of_parents of
```

```
2: begin
 j := 1;
 repeat
  mate1 := select (p, number_of_members, sumfitness);
  mate2 := select (p, number_of_members, sumfitness);
  crossover2(p^.pop[mate1].bits, p^.pop[mate2].bits, q^.pop[j].bits,
           q^.pop[j + 1].bits, number_of_bits, pcross, pmutation,
           cross_type);
  j := j + 2;
 until j > number_of_members;
   end;
3: begin
 j := 1;
 repeat
  mate1 := select (p, number_of_members, sumfitness);
  mate2 := select (p, number_of_members, sumfitness);
  mate3 := select (p, number_of_members, sumfitness);
  if cross_type = 's' then
  begin
    crossover6(p^.pop[mate1].bits, p^.pop[mate2].bits, p^.pop[mate3].bits,
          q^.pop[j].bits, q^.pop[j + 1].bits, q^.pop[j + 2].bits,
          q^.pop[j + 3].bits, q^.pop[j + 4].bits, q^.pop[j + 5].bits,
          number_of_bits, pcross, pmutation, cross_type);
  j := j + 6;
  end
  else begin
  crossover3(p^.pop[mate1].bits, p^.pop[mate2].bits, p^.pop[mate3].bits,
          q^.pop[j].bits, q^.pop[j + 1].bits, q^.pop[j + 2].bits,
          number_of_bits, pcross, pmutation, cross_type);
  j := j + 3;
  end;
 until j > number_of_members;
 end; {case number 3}
 end; {case}
 evaluate (q, number_of_members, number_of_bits);
 temp_ptr := p;
 p := q;
 q := temp_ptr;
end;

{  The following procedure is used to keep track of the on-line and        }
{  off-line averages.  It also keeps track of the best individual found    }
{  for a given trial.                                                        }

procedure stats (var p: pop_ptr; var best_value, avg, max: real;
                var sumfitness : real;
```

```
               var best_bits : bit_string; number_of_members: integer;
               var online_sum, online_average, offline_sum,
               offline_average: real);

var j : integer;
    sum_realfitness : real;
begin
 sumfitness := p^.pop[1].fitness;
 sum_realfitness := p^.pop[1].real_fitness;
 best_value := p^.pop[1].real_fitness;
 best_bits := p^.pop[1].bits;
 max := p^.pop[1].fitness;
 for j := 2 to number_of_members do with p^.pop[j] do
 begin
  sumfitness := sumfitness + fitness;
  sum_realfitness := sum_realfitness + real_fitness;
  if real_fitness < best_value then
   begin
    best_value := real_fitness;
    best_bits := bits;
   end;
  if fitness > max then max := fitness;
 end;
 avg := sum_realfitness/number_of_members;
 online_sum := online_sum + sum_realfitness;
 online_average := online_sum / (number_of_members*(gen + 1.0));
 offline_sum := offline_sum + best_value;
 offline_average := offline_sum / (gen + 1.0);
end;

{  The following procedure was used during the debugging phase.  It        }
{  outputs the bit string value of a particular population member.         }

procedure writechrom (chrom: bit_string; number_of_bits:integer);
var j : integer;
begin
 for j := number_of_bits downto 1 do
  if chrom[j] then write ('1')
   else write ('0');
end;

{  The following procedure was used during the debugging phase.  It        }
{  outputs various metrics used to measure performance.                    }

procedure report (gen:integer; best_value, avg, online, offline : real);
var j : integer;
```

```
begin
 writeln (outfile, 'generation ',gen:4,' min = ',best_value:6:4,' on = ',
         online:6:4,' off= ',offline:6:4);
end;

{  The following procedure gets the input from the data file.                    }

procedure get_input (var maxgen, function_number, number_of_parents,
                     number_of_members: integer;
                     var blank_space, cross_type: stringone;
                     var pcross, pmutation : real);
begin
readln (infile, maxgen, function_number, number_of_parents,
       number_of_members, blank_space, cross_type, pcross, pmutation);
end;

{  The following procedure keeps running totals used for on-line and             }
{  averages.                                                                      }

procedure add_totals (gen : integer; online_average, offline_average : real);
begin
totals [1,gen] := totals [1,gen] + online_average;
totals [2,gen] := totals [2,gen] + offline_average;
end;

begin  { main program }
{ openfile   (infile, 'genalg.in');   required for Turbo Pascal I/O }
 reset (infile, 'genalg.in');
{  The following values are used in function 5 }
a[1,1] := -32.0;
a[1,2] := -16.0;
a[1,3] := 0.0;
a[1,4] :=  16.0;
a[1,5] :=  32.0;
a[1,6] := -32.0;
a[1,7] := -16.0;
a[1,8] := 0.0;
a[1,9] :=  16.0;
a[1,10] := 32.0;
a[1,11] := -32.0;
a[1,12] := -16.0;
a[1,13] := 0.0;
a[1,14] := 16.0;
a[1,15] := 32.0;
a[1,16] := -32.0;
a[1,17] := -16.0;
```

```
a[1,18] := 0.0;
a[1,19] := 16.0;
a[1,20] := 32.0;
a[1,21] := -32.0;
a[1,22] := -16.0;
a[1,23] := 0.0;
a[1,24] := 16.0;
a[1,25] := 32.0;

for k := 1 to 5 do
  a[2,k] := -32.0;
for k := 6 to 10 do
  a[2,k] := -16.0;
for k := 11 to 15 do
  a[2,k] := 16.0;
for k := 16 to 20 do
  a[2,k] := 32.0;
for k := 21 to 25 do
  a[2,k] := 0.0;

get_input (maxgen, function_number, number_of_parents, number_of_members,
           blank_space, cross_type, pcross, pmutation);

rewrite (outfile, 'ga.out');

{  The following loop goes from the initial pcross value (usually 0.6)            }
{  in increments of 0.1 (stopping at 0.9).                                        }

for jj := 1 to 4 do
begin
 seed := 25.0;
 writeln (outfile, 'Maximum number of generations ',maxgen);
 writeln (outfile, 'Function number ', function_number);
 writeln (outfile, 'Number of parents ', number_of_parents);
 writeln (outfile, 'Number of population members ', number_of_members);
 writeln (outfile, 'Probability of crossover ',pcross:6:4);
 writeln (outfile, 'Probability of mutation ', pmutation:6:4);
 writeln (outfile, 'Random seed ', seed:8:2);
 for j := 1 to 2 do
   for k := 0 to maxgen do
     totals [j,k] := 0.0;
 for m := 1 to number_of_trials do
 begin
 new (p);
 new (q);
 { Initialize the required values for the function under consideration. }
```

```
case function_number of
1 : begin
     number_of_bits := 30;
     f_max := 78.3363;
     number_of_genes := 3;
     denominator := 100.0;
     f_max_addition := 0.0;
    end;
2 : begin
     number_of_bits := 24;
     f_max := 3905.9263;
     number_of_genes := 2;
     denominator := 1000.0;
     f_max_addition := 0.0;
    end;
3 : begin
     number_of_bits := 50;
     f_max := 50.0;
     number_of_genes := 5;
     denominator := 100.0;
     f_max_addition := 0.0;
    end;
4 : begin
     number_of_bits := 240;
     f_max := 2430.0;
     number_of_genes := 30;
     denominator := 100.0;
     f_max_addition := 12.0;
    end;
5 : begin
     number_of_bits := 32;
     f_max := 3.82;
     { this is approx. the max. possible function value}
     number_of_genes := 2;
     denominator := 1000.0;
     f_max_addition :=  0.0;
    end;
end;
online_sum := 0.0;
offline_sum := 0.0;
initialize (p, number_of_members, number_of_bits);
gen := 0;
stats (p, best_value, avg, max, sumfitness, best_bits, number_of_members,
      online_sum, online_average, offline_sum, offline_average);
global_best_value := best_value;
global_best_gen := gen;
```

```
 add_totals (gen, online_average, offline_average);
 repeat
  gen := gen + 1;
  generation (p, number_of_parents, number_of_members, number_of_bits,
             pcross, pmutation, sumfitness, cross_type);
  stats (p, best_value, avg, max, sumfitness, best_bits, number_of_members,
        online_sum, online_average, offline_sum, offline_average);
  if best_value < global_best_value then
   begin
    global_best_value := best_value;
    global_best_gen := gen;
   end;
  add_totals (gen, online_average, offline_average);
  if (gen mod 2) = 0 then f_max := max + f_max_addition;
 until (gen >= maxgen);
 dispose (p);
 dispose (q);
 best_of_trial [m] := global_best_value;
 best_of_gen [m] := global_best_gen;
end;
 writeln (outfile, 'online average ',' offline average');
 for j := 1 to 2 do
   for k := 1 to maxgen do
     totals [j,k] := totals[j,k] / number_of_trials;
 for k := 1 to maxgen do
     writeln (outfile, totals [1,k]:12:10, '  ', totals[2,k]:12:10);
 for k := 1 to number_of_trials do
   begin
     write (outfile, 'trial ',k,'  ',best_of_trial [k]:12:10);
     writeln (outfile, '  during generation ',best_of_gen[k]:4);
   end;
pcross := pcross + 0.1;
end;
 close (outfile);
end.
```

VITA

Lawrence Vincent Edmondson was born September 24, 1961 in Independence, Missouri. He received his primary and secondary education in Independence, Missouri.

In May 1983 he received a Bachelor of Science degree in Computer Science and Mathematics from Central Missouri State University in Warrensburg, Missouri, graduating *magna cum laude*. In July 1985 he received a Master of Science degree in Computer Science from the University of Missouri-Rolla, in Rolla, Missouri. Following his graduation, Vince was employed in the research and development laboratories of AT&T in Middletown, New Jersey from 1985 to 1987.

In pursuit of the Ph.D. in Computer Science, he returned to the University of Missouri-Rolla in August 1987. While at Rolla, he held a Chancellor's Fellowship and a graduate teaching assistantship. Upon obtaining ABD status in 1990, Vince accepted his current position of Assistant Professor with the Department of Mathematics and Computer Science at Central Missouri State University.