An Empirical Study of Genetic Operators in Genetic Algorithms

Xin Yao[a*†]

[a]Department of Computer Science, University College, University of New South Wales
Australian Defence Force Academy, Canberra, ACT 2600, Australia

Genetic algorithms are multi-agent search strategies applicable to a wide range of problems. However, it is often very difficult in practice to design an optimal set of genetic operators for a problem, because a genetic operator which works well for a problem might not work well for another problem. This paper tries to understand when and why some genetic operators are useful and how we can combine them together to improve the performance of genetic algorithms. Experiments have been carried out to analyse the role of crossover and mutation as well as selection mechanisms. Several genetic algorithms with different genetic operators and selection mechanisms are used in the empirical study. The study suggests that "greedy" crossover and "hard" selection with a low mutation rate often give genetic algorithms better performance.

1. INTRODUCTION

Two distinct features of genetic algorithms (GAs) are the multi-agent search strategy and the information exchange among agents. Such information exchange is achieved by genetic operators and selection mechanisms. Genetic operators use the information contained in parental solutions to produce hopefully better child solutions. Selection mechanisms are used to discriminate among solutions generated by different agents so that better solutions are kept and poor ones are eliminated. It is, unfortunately, not well understood which genetic operators and selection mechanisms perform best in practice. The aim of this paper is to gain such understanding through an empirical study.

The traveling salesman problem (TSP) has been used extensively in evaluating different genetic operators and selection mechanisms in GAs [1, 2]. The TSP is chosen because "it captures in a simple, elegant way many of the open GA issues" [3]. It has been identified as an example where a good set of genetic operators is hard to find. Moreover, the TSP is a well-known NP-complete problem. The study of the TSP would have a wide implication because of mutual reductions among NP-complete problems. Hence, this paper will concentrate on comparing different GAs for solving the TSP. The TSP is regarded as a common bench mark for comparison in this paper.

The TSP can be formulated as follows. Given N cities, $1, 2, \cdots, n$, and distances, $d_{ij}, 1 \leq i, j \leq n$, among them, find a permutation, $(x_1, x_2, \cdots, x_N)$, of $(1, 2, \cdots, n)$ such that $D = \sum_{i=1}^{N} d_{x_i x_{i+1}}$ is minimum, where $x_{N+1} = x_1$. That is, find a shortest tour of N cities by visiting each city exactly once and returning to the initial city.

GAs can be summarised as Figure 1. They start with a population of randomly generated solutions, i.e., generation $G(0)$. In generation $G(i), i \geq 0$, parental solutions are selected based on their fitness or ranks in the population. Then genetic operators, such as crossover, mutation, inversion, etc., are applied to the parental solutions to produce the next generation $G(i+1)$. This process continues until a pre-defined termination condition is satisfied. In Figure 1, the set of genetic operators can vary greatly, e.g., it may include crossover, mutation and inversion; it may include only crossover or only mutation. So GAs in this paper are defined in a general sense. They stand for a class of algorithms with different genetic operators and selection mechanisms.

There have been some attempts in solving the TSP by GAs [1, 2, 4]. Various recombination (crossover) operators have been proposed, but systematic comparisons among them are few. This paper compares six GAs and some of their

*Email: xin@csadfa.cs.adfa.oz.au.
†Published in **Microprocessing and Microprogramming**, Vol. 38, pp.707–714, 1993.

1. Generate the initial population $G(0)$ at random, and set $i = 0$;
2. REPEAT
   (a) Select parents from $G(i)$ based on their fitness or ranks in $G(i)$;
   (b) Apply genetic operators to the parents and get generation $G(i+1)$;
   (c) $i = i + 1$;
3. UNTIL 'termination criterion' satisfied

Figure 1. Framework of GAs.

variants under the same experimental setup in order to identify each GA's advantages and disadvantages. The paper also discusses the issue of when to use what genetic operators in GAs. Three performance metrics are adopted in the paper; the best tour length, the average tour length in a population, and the entropy in a population, which is used as a measurement of population diversity [1]. The empirical study in this paper shows that some GAs can keep high population diversity for a long time during genetic search, thus can offer improvement of tour lengths for a long time. But such improvements tend to be very slow. A better approach is to adopt "greedy" genetic operators and "harder" selection mechanisms in GAs so that GAs can converge to a near optimal tour length quickly. Mutation can be introduced in this fast converging process to keep population diversity at a relatively high level.

This paper is organised as follows. Section 2 describes GAs compared. Section 3 gives the criteria used to compare them. Section 4 presents our experimental findings. Finally, Section 5 concludes with a summary of the paper and future research directions.

2. ALGORITHMS COMPARED

Six different GAs are considered in this paper.

GA1 is based on Whitley *et al.*'s algorithm [2]. The only recombination operator used in GA1 is described in Figure 2, which introduces as few new edges into the child tour as possible in order to inherit edges and subtours in the parental tours. As indicated by Whitley *et al.* [2], "an ideal operator should construct an offspring tour by exclusively using links present in the two parent structures." But this is difficult to achieve. Two kinds of implicit mutation are introduced in GA1's recombination operator [2]. One is caused by the edge from the final city to the initial city, the other is caused by "omission," i.e., an edge present in both parents is not passed to the offspring.

GA2 is based on Grefenstette's algorithm [1]. Its recombination operator can also be described by Figure 2, but with a different Step 2d. Step 2d in GA2 is probabilistically greedy. The next city is not chosen according to the number of entries in the edge list, but chosen according to the distances from the current city. That is, city x_j will be chosen according to probability $p_{x_j}, 1 \leq j \leq 4$,

$$p_{x_j} = \frac{1/d_{x_i x_j}}{\sum_{l=1}^{4}(1/d_{x_i x_l})} \tag{1}$$

where x_i is the current city and $d_{x_i x_j}$ the distance between x_i and x_j, $x_i \neq x_j$. Although GA2 adopts a recombination operator described in [1], it is different from GAs described in [1] because it does not use any hueristics to generate the initial population or to improve the final tours.

GA3 is a variant of GA1. It is different from GA1 only in Step 2d where ties are broken according to the distance of the edges, rather than at random. For example, if $k(\leq 4)$ cities have the same number of entries in their edge lists, then city x_j will be chosen according to probability $p_{x_j}, 1 \leq j \leq k$,

$$p_{x_j} = \frac{1/d_{x_i x_j}}{\sum_{l=1}^{k}(1/d_{x_i x_l})} \tag{2}$$

where x_i is the current city and $d_{x_i x_j}$ the distance between x_i and x_j, $x_i \neq x_j$.

1. Construct an edge map which contains all the different edges from the two parental tours. For example, if the two parental tours are (a, b, c, d, e) and (b, e, d, c, a), then the edge map is constructed as:

   a : b, c, e

   b : a, c, e

   c : a, b, d

   d : c, e

   e : a, b, d

2. Construct a child tour from the edge map.

   (a) Choose the initial city at random and record it as the current city.

   (b) Mark the current city "visited" and remove all its occurrences from the right-hand side (called the edge list) of the edge map.

   (c) Go to the next step if the current city has entries in its edge list, or else go to Step 2e.

   (d) Determine which of the cities in the edge list of the current city has the fewest entries in its own edge list. The city with the fewest entries becomes the current city. Ties are broken randomly. Go to Step 2b.

   (e) Stop if all cities have been visited, or else randomly choose an unvisited city as the current one and go to Step 2b.

Figure 2. Recombination operator used in GA1.

Although the above three recombination operators include implicit mutation, they are in essence crossover operators. They all use bisexual recombination of two parents. To investigate whether or not asexual operators perform better than bisexual ones, three more GAs are considered. As indicated at the beginning of this paper, the term GAs are used in a general sense. Some researchers do not consider asexual operators alone are sufficient to define a GA, i.e., GAs should have bisexual recombination operators. GAs with only mutation are similar to evolutionary programming [5] and evolution strategies [6], but the level at which operators are applied to individuals is different.

GA4, GA5, and GA6 all have the same mutation operator described as follows. Given a parent tour $t_p = (x_1, x_2, \cdots, x_N)$, where N is the number of cities. A child tour t_c is generated by randomly choosing two cities in t_p and reversing the subtour between the two cities.

The only difference among GA4, GA5, and GA6 is their selection mechanisms. GA4's selection mechanism is based on the competition method described in [7]. Given a population, $G(i)$, of P solutions (tours), generate P offsprings by mutation. Each of the $2P$ solutions competes against 10% of the $2P$ which are chosen at random. (A solution does not compete against itself.) The probability of attaining a win over the opponent is the opponent's fitness divided by the sum of the two competing solutions' fitness. For example, if solution i's fitness is 100 and solution j's fitness is 200, then solution i wins with probability 2/3. In GA4, the next generation, $G(i+1)$, consists of P solutions with most wins in the competition. (Note that fitness in our experiments is defined as the inverse of the tour length.)

GA5 uses the fitness-based selection mechanism. Let the fitness of a solution j in a population be f_j, $1 \leq j \leq M$, then solution j's reproduction probability is

$$P_{GA5}(j) = \frac{f_j}{\sum_{k=1}^{M} f_k} \tag{3}$$

where M is the population size.

GA6 adopts rank based selection mechanism. In GA6, tours in a population are first sorted

in a non-descending order according to their lengths. Let the M sorted tours be numbered as $0, 1, \cdots, M-1$. Then the $M-j$th tour is selected with probability

$$P_{GA6}(M-j) = \frac{j}{\sum_{k=1}^{M} k} \quad (4)$$

3. PERFORMANCE METRICS

Three major performance metrics used in our comparison are the best tour length, the average tour length in a population and the entropy in the population. The entropy of a population is used to measure population diversity and is defined as [1]

$$H = \frac{1}{N}\sum_{i=1}^{N} H_i$$

where N is the number of cities and

$$H_i = -\frac{1}{\log(N)}\sum_{j=1}^{N}\left(\frac{n_{ij}}{2P}\log\left(\frac{n_{ij}}{2P}\right)\right)$$

P is the population size. n_{ij} is the number of edges connecting city i and city j in the population. It is clear that H approaches 0 when the population converges, and attains the maximum when the population diversity is the largest. Although the definition of entropy here is not the same as the standard definition, it is good enough to measure the population diversity.

4. EXPERIMENTAL RESULTS

Ten problem instances of the Euclidean plane TSP with 100 cities are used in our experiments. They are generated by randomly choosing 100 city locations in a square region so that the expected minimum tour length L^* is 100. L^* is determined by the following formula [1]:

$$L^* = K\sqrt{NX^2}$$

where X is the length of a side of the square, N is the number of cities, and K is an empirical constant of 0.765 [1]. Bonomi and Lutton [8] used $K = 0.749$ in their paper. We use $K = 0.765$ given in [1]. Since the aim of our paper is to compare different GAs, not to find an optimal solution by a particular GA, this choice is not crucial as long as all GAs compared use the same K. That is, the length of a side of the square is

$$X = \frac{100}{0.765\sqrt{100}} \approx 13.0719$$

For GAs compared in this paper, the following parameters are set to be the same in order to evaluate different genetic operators and selection mechanisms: the same population size, 100; the same initial population, the same probability of recombination (for GA1, GA2, and GA3), 0.7; the same rank-based selection (for GA1, GA2, GA3, GA6); and the same mutation operator (for GA4, GA5, and GA6).

4.1. Experiment 1

The first experiment is aimed to investigate the impact of greedy recombination operators on genetic search. GA1, GA2, and GA3 are compared with each other. The only difference among them is Step 2d in Figure 2. All the other parameters are kept the same, as indicated before. Typical runs of GA1, GA2, and GA3 for 200 generations on a TSP instance are shown in Figure 3 to 5. The three figures also contain results of GA4, GA5, and GA6, which will be explained in Section 4.2. The average results over ten TSP instances are summarised in Table 1. It is clear that GA2 performed best among the three GAs in terms of tour lengths. It also maintained a reasonable level of population diversity. GA1 maintained a very high level of population diversity, which indicates its great potential in improving solutions further. But the improvement is very slow. Table 2 shows the result, averaged over ten TSP instances, of GA1 after 4000 generations.

This experiment suggests that GA2, which has a probabilistically greedy recombination operator, is most suitable for finding a near optimal solution within a short time, while GA1 might be more suitable for solving problems where the computational cost is not the key issue. As far as GA3 is concerned, although it found better solutions than those found by GA1 after 200 generations, it lost its ability in further improving its solutions because of its low population diversity.

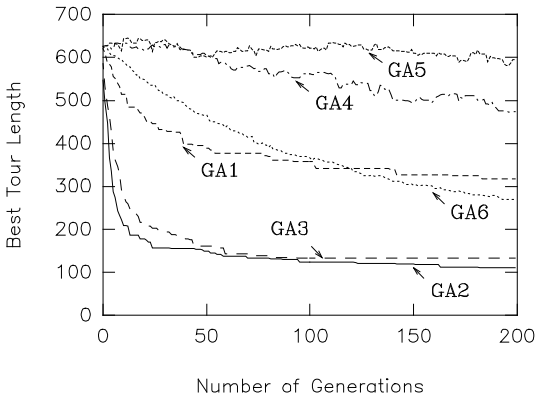

Figure 3. Typical runs of six GAs for 200 generations on a TSP instance. Best tour lengths are shown.

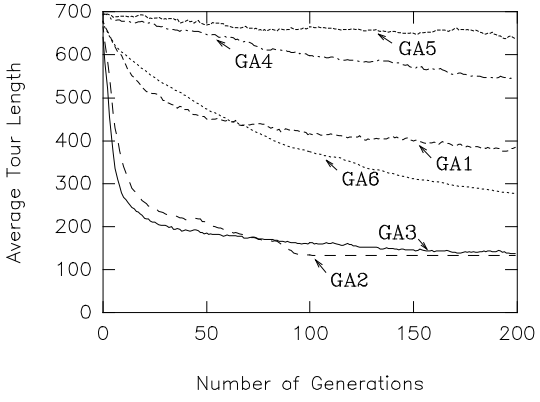

Figure 4. Typical runs of six GAs for 200 generations on a TSP instance. Average tour lengths are shown.

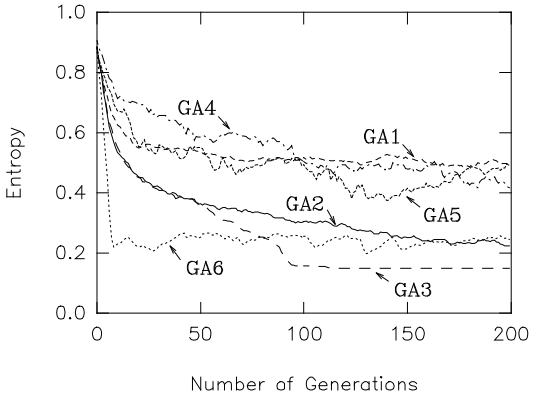

Figure 5. Typical runs of six GAs for 200 generations on a TSP instance. Entropies are shown.

	Mean	Std Dev	Std Err
GA1 (Best)	309.59	13.535	4.280
GA2 (Best)	110.48	4.548	1.438
GA3 (Best)	126.07	5.025	1.589
GA1 (Average)	387.03	12.105	3.828
GA2 (Average)	127.26	11.465	3.626
GA3 (Average)	126.67	6.260	1.980
GA1 (Entropy)	0.487	0.015	0.005
GA2 (Entropy)	0.200	0.028	0.009
GA3 (Entropy)	0.151	0.000	0.000

Table 1
Average results, over ten TSP instances, of running three GAs for 200 generations, where "Best" indicates the best solution found, "Average" indicates the average tour length in the last generation, and "Entropy" indicates the entropy of the last generation. *Mean, Std Dev*, and *Std Err* in the table stand for the mean, standard deviation, and standard error respectively.

	Mean	Std Dev	Std Err
GA1 (Best)	239.56	11.643	3.682
GA1 (Average)	338.38	21.014	6.645
GA1 (Entropy)	0.414	0.308	0.010

Table 2
Average results of running GA1 for 4000 generations.

In fact, GA3 converged prematurely within first 120 generations and never improved its solutions afterwards for all ten TSP instances.

This experiment also shows that the population diversity does not necessarily decrease as a more greedy genetic operator is used. Step 2d of GA2 is apparently more greedy than Step 2d of GA3 since GA2 biases the shortest edge among all four edges in two parental tours, while GA3 only biases a shorter one when there is a tie. But GA2 still maintains higher population diversity than GA3 does.

It is well-known that GAs are good at "coarse-grained" search, i.e., locating promising regions in a search space, but not well suited for "fine-grained" search, i.e., fine-tuning individual solutions in a good region. Greedy genetic operators offer a way to introduce such fine-tuning ability into genetic search. In GA1, fitness is only considered during selection. The recombination operator does not consider fitness of offsprings. But two good parents can generate poor offsprings because not every part of the parents is good unless the parents are global optima. It would be more efficient to avoid generating poor offsprings than to depend merely on selection to weed out poor solutions. In a sense, greedy genetic operators provides a secondary selection pressure against poor solutions. This pressure is generated by evaluating only part of a solution, e.g., edges, so we call it local selection pressure. The (global) selection pressure is generated by evaluating whole solutions.

4.2. Experiment 2

There are two aims of Experiment 2. The first is to compare the recombination operators used in GA1, GA2, and GA3 with the mutation operator used in GA4, GA5, and GA6. The second aim is to compare competition, fitness-based selection, and rank-based selection used in GA4, GA5, and GA6 respectively. All the conditions except for those compared are set the same in the experiment. Typical runs of GA4, GA5, and GA6 for 200 generations on a TSP instance are shown in Figure 3 to 5. Table 3 gives the average results, over ten TSP instances, of running them for 200 generations.

	Mean	Std Dev	Std Err
GA4 (Best)	479.81	24.825	7.850
GA5 (Best)	565.96	15.303	4.839
GA6 (Best)	264.40	5.607	1.773
GA4 (Average)	539.08	25.351	8.017
GA5 (Average)	640.35	21.636	6.842
GA6 (Average)	274.97	5.798	1.834
GA4 (Entropy)	0.466	0.053	0.017
GA5 (Entropy)	0.469	0.030	0.010
GA6 (Entropy)	0.232	0.010	0.003

Table 3
Average results of GA4, GA5 and GA6 over ten TSP instances. The number of generations is 200.

Comparing this experiment with the previous one, it is found that GA2's recombination operator can offer faster convergence than GA6's mutation operator can since the only difference between them is the genetic operator. But GA6 has slightly higher population diversity which indicates its higher potentiality in further improving its solutions if more computation time is allowed. GA4 and GA5 have similar behaviour to GA1's because they all have poor solutions and high population diversity. They all converge very slowly. Table 4 gives the results of running GA4, GA5, and GA6 for 4000 generations.

The good performance of GA6 lies in its high selection pressure produced by rank-based selection. GA2 performs better than GA6 because it has local selection pressure produced by the recombination operator in addition to the global selection pressure. GA4 and GA5 converge very

	Mean	Std Dev	Std Err
GA4 (Best)	383.563	14.451	4.570
GA5 (Best)	511.252	10.544	3.334
GA6 (Best)	117.766	4.194	1.326
GA4 (Average)	486.419	12.616	3.990
GA5 (Average)	633.532	30.306	9.584
GA6 (Average)	128.252	5.273	1.667
GA4 (Entropy)	0.443	0.033	0.010
GA5 (Entropy)	0.437	0.036	0.012
GA6 (Entropy)	0.196	0.006	0.002

Table 4
Average results of GA4, GA5 and GA6 over ten TSP instances. The number of generations is 4000.

slowly because the selection pressure produced by competition or fitness-based selection is not high enough. GA4 can be further analysed as follows.

Given a parent tour $t_p = (x_1, x_2, \cdots, x_N)$, where N is the number of cities. A child tour t_c is generated by randomly choosing two cities in t_p and reversing the subtour between the two cities. Let the two cities be x_i and x_j, then the difference in tour length between the parent and the child is

$$\begin{aligned} \Delta L &= L_c - L_p \\ &= d_{x_{i-1}x_j} - d_{x_{i-1}x_i} + d_{x_i x_{j+1}} - d_{x_j x_{j+1}} \end{aligned}$$

where d's are distances between cities. L_p and L_c are tour lengths of the parental tour t_p and the child tour t_c respectively. Hence,

$$|\Delta L| \leq 2 * \max\left\{d_{x_{i-1}x_j}, d_{x_{i-1}x_i}, d_{x_i x_{j+1}}, d_{x_j x_{j+1}}\right\}$$

Suppose t_c is competing against t_p, then the probability of attaining a win by t_c is P_{cp},

$$P_{cp} = \frac{L_p}{L_p + L_c} = \frac{L_c - \Delta L}{2L_c - \Delta L} \tag{5}$$

Because ΔL is usually very small in comparison with L's when N is large, an improvement of ΔL over L_p only results in a winning probability slightly larger than 0.5 according to (5). This is why GA4 has so many difficulties in moving towards better solutions.

In general, let $t_1, t_2, \cdots, t_M$ be M opponents of t_i. Then the expected number of wins of t_i is

$$W_i = \sum_{j=1}^{M} P_{ij} = \sum_{j=1}^{M} \frac{L_i + \Delta L_{ij}}{2L_i + \Delta L_{ij}} \tag{6}$$

where $\Delta L_{ij} = L_j - L_i$. Obviously, a small ΔL_{ij} will not change W_i a lot. Since all the tours are either generated at random initially or generated by mutation, the difference among W_i's is quite small unless there are tours which are much better than others in the initial population. Generation of a tour with a very long or short tour length by mutation is only possible when there are edges which differ greatly. This is certainly not the case in our experiments, where TSP instances are generated uniformly at random.

The above analysis and experimental results demonstrate the importance of selection pressure in genetic search. The advantage of recombination operators lie in their ability to provide extra local pressure against poor solutions. If a GA without recombination operators can produce enough selection pressure against poor solutions, its performance would be as good as GAs with recombination operators, or even better. The results of GA1 and GA6 illustrate this point.

4.3. Experiment 3

Although GAs with both crossover and mutation operators are quite common, few attempts have been made to use such GAs in solving the TSP. In our previous experiments, GA1, GA2, and GA3 use a crossover operator, while GA4, GA5, and GA6 use a mutation operator. In this experiment, the mutation operator used in GA4, GA5, and GA6 is added to GA1, GA2, and GA3. The mutation probability is set to 0.002. We also tried several other mutation probabilities, i.e., 0.001, 0.003, 0.005, and 0.01. The probability of 0.002 seemed to give the best results. Table 5 shows the average results, over ten TSP instances, of running GA3 for 200 generations. The improvement over the results of GA3 in Table 1 is apparent. Similar improvement has also been observed for GA1 and GA2. But the improvement of GA1 is little because it already had high population diversity. An additional mutation operator did not help much. Better results

	Mean	Std Dev	Std Err
Best	121.84	6.006	1.899
Average	126.24	5.605	1.773
Entropy	0.162	0.002	0.001

Table 5
Average result of GA3, over ten TSP instances, with mutation probability 0.002. The number of generations is 200.

were obtained by running the three GAs for 4000 generations.

This experiment also confirms that premature convergence can be tackled by introducing a mutation operator. While this finding is well-known for many problems, it is probably the first time (to our best knowledge) that both crossover and mutation are applied to solving the TSP.

5. CONCLUSION

An empirical study of genetic algorithms using the TSP is carried out in this paper. Three experiments have been performed. The first experiment compares different recombination operators used in solving the TSP. It is found that greedy recombination operators can offer good solutions within a relatively short time. It is also shown, by comparing GA2 with GA3, that a more greedy operator does not necessarily mean the faster decrease of population diversity.

The second experiment compares recombination operators with mutation operators and compares different selection mechanisms. In general, greedy recombination operators (in GA2) tend to perform better than mutation operators (in GA4, GA5, and GA6). But some recombination operators (in GA1) perform poorer than mutation operators. The concept of local selection pressure is introduced to explain why greedy recombination operators perform better. It is demonstrated that selection pressure, both local and global, is a key factor in obtaining a near optimum quickly. An analysis of the competition mechanism used in [7] is also given, which explains why the algorithm described in [7] is slow. An improvement to this algorithm, GA6, has been shown to have much better performance.

The third experiment demonstrates the advantage of applying both recombination and mutation operators to the TSP. This experiment shows that use of both recombination and mutation in solving the TSP can result in better solutions while keeping high population diversity.

Further work to be done along the line of this paper includes a more detailed analysis of various selection mechanisms and a comparative study of GAs and multiple simulated annealing.

REFERENCES

1. J. J. Grefenstette, "Incorporating problem specific knowledge into genetic algorithms," In L. Davis (ed.), *Genetic Algorithms and Simulated Annealing*, pp.42–60, Morgan Kaufmann, San Mateo, CA, 1987.
2. D. Whitley *et al.*, "The traveling salesman and sequence scheduling: quality solutions using genetic edge recombination," In L. Davis (ed.), *Handbook of Genetic Algorithms*, pp.350–372, Van Nostrand Reinhold, New York, NY, 1991.
3. K. De Jong, "Genetic algorithms: a 10 year perspective," In *Proc. of ICGA-85*, pp.169–177, 1985.
4. J. J. Grefenstette *et al.*, "Genetic algorithms for the traveling salesman problem," In *Proc. of ICGA-85*, pp.160–168, 1985.
5. L. J. Fogel *et al.*, *Artificial Intelligence Through Simulated Evolution.* John Wiley & Sons, New York, NY, 1966.
6. H.-P. Schwefel, *Numerical Optimization of Computer Models*, John Wiley & Sons, 1981.
7. D. B. Fogel, "An evolutionary approach to the traveling salesman problem," *Biological Cybernetics*, 60:139–144, 1988.
8. E. Bonomi and J.-L. Lutton, "The N-city traveling salesman problem: statistical mechanics and the Metropolis algorithm," *SIAM Review*, 26:551–568, 1984.