# Altering Genetic Algorithm Parameters for the Traveling Salesman Problem: An Empirical Study

Travis Zimmerman
Department of Computer Science
University of West Florida
Pensacola, FL
(850) 225-0143
tez1@students.uwf.edu

John W. Coffey
Department of Computer Science
University of West Florida
Pensacola, FL
(850) 471-3183
jcoffey@uwf.edu

## ABSTRACT
Optimization problems are an important area of research in computer science, and genetic algorithms are becoming popular methods for solving these problems. This paper offers a brief introduction to optimization problems as well as genetic algorithms. An empirical study of genetic algorithms for the traveling salesman problem is presented, showing that genetic algorithms are effective at solving optimization problems.

## Categories and Subject Descriptors
I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search – *Graph and tree search strategies*

## General Terms
Algorithms, Performance, Experimentation

## Keywords
Genetic algorithms, Evolutionary algorithms, Evolutionary computing, Traveling Salesman Problem

## 1. INTRODUCTION
The concept of natural selection has inspired computer scientists to adapt mechanisms from biology to solve complex problems. A genetic algorithm (GA) begins with a random population of possible solutions to a problem. The algorithm then produces successive generations of solutions, evolving toward a final solution. As in natural selection, the fittest solutions in any given generation are more likely to mate and pass on their genetic material. A GA can evolve a nearly optimal solution in a reasonable amount of time.

After describing optimization problems and GAs, a case study in the use of a GA to solve an intractable problem will be presented. The study compares run times involved in brute-force solutions with those achieved by the GA. Additionally, the quality of the solutions that the GA produces as several parameters are manipulated will be discussed.


*GECCO'06*, July 8–12, 2006, Seattle, WA, USA.


## 2. OPTIMIZATION & GENETIC ALGORITHMS
Many problems that have multiple solutions are too computationally complex to find the best solution in a reasonable amount of time. It is often more practical to use a heuristic approach to find a solution that might not be the best, but is good enough. This is the problem of optimization. Optimization is the process of adjusting inputs to a device, experiment, or process to find the minimum or maximum output [1]. Because finding the optimal solution involves minimizing or maximizing a quantity, optimization problems are often treated as minimization problems. If the objective is to maximize a quantity, it is easy enough to negate the process of finding a minimum.

In mathematical terms, a certain function, commonly called the cost function, must be minimized across an interval. This interval is known as the search space. Searching the search space for the lowest output of the cost function will yield the minimum quantity. However, it is not possible to conduct an exhaustive search for problems with search spaces that grow exponentially or worse, hence the need for creative algorithms. There are several current methods for solving optimization problems, such as hill-climbing [2], simulated annealing [2], beam searches [2], and GAs [1].

GAs are based on the principles of natural selection. A GA begins with a random set of possible solutions to the problem (an optimization problem has many solutions; some are better than others). The set of solutions is called the population. Each solution is evaluated by a fitness function, and the best solutions have a higher chance of mating to produce the next population. The GA repeats the process of mating to form new populations until an acceptable solution is found, predetermined criteria are met, or all the solutions become the same and further execution will not yield new solutions (convergence).

Random mutations can also introduce diversity (new solutions) into each population. GAs can be highly customized depending on the problem. GAs can use strings of bits for basic satisfiability problems or permutations of digits or letters for permutation problems.

## 3. TRAVELING SALESMAN PROBLEM
The traveling salesman problem (TSP) is a well-known problem in computer science. A salesman must visit several different cities in any order and return to the starting point. The cities should be visited in the order that minimizes the total distance traveled to minimize the time and cost of the journey.

The simplest way to find the optimal solution is to use a brute force algorithm, which computes the cost of every possible journey (tour) through all the cities, and reports the minimum. The brute force method is a guaranteed way to find the optimal tour, however it is unreasonable to use in practice because of the TSP's computational complexity. If the *n* cities are represented as a list of *n* different digits, then the order the salesman visits the cities would be the order of the digits. The brute force algorithm would thus have to attempt every possible permutation (ordering) of these digits. If there are *n* digits, there are n-factorial (*n!*) ways to permute the list of digits. Factorially growing search spaces require an unreasonably long amount of time for a computer to find an optimal solution. If there are only twenty cities, the number of possible tours to evaluate is 2,432,902,008,176,640,000. Assuming a computer can evaluate one possible tour every second (which is not unreasonable), it would still take over 77 billion years to find the optimal solution – for only twenty cities!

Obviously the brute force method is not acceptable for the TSP. A GA, however, can deliver a solution in a reasonable amount of time. The solution is not guaranteed to be optimal, but generally it is close enough.

The TSP is a permutation problem – the optimal solution is the optimal ordering of the cities involved. A few precautions are required to ensure the crossover and mutation operators generate valid tours. The crossover operator cannot exchange cities in a haphazard way because some cities would be deleted and others would be repeated. Similarly, the mutation operator cannot simply change a random city, because this could result in deletion or repetition of a city. Both operators must be carefully designed so that they do not invalidate any tours.

The TSP is a popular and useful way to examine the possibilities and effectiveness of GAs. The TSP is used in the empirical study described in the next section.

# 4. TSP EMPIRICAL STUDY

A brute force and a genetic algorithm were both programmed to conduct an analysis of GAs for the TSP. The brute force program was used to find the minimum tour lengths for small numbers of cities, which were then compared against the results of the GA.

The GA used the crossover operator proposed by [3], the partially-mapped crossover (PMX) operator. First, PMX chooses two positions at random, which identifies the substrings that will be swapped. The substring from the first tour is mapped to the second by a series of swaps, and the same process maps the substring from the second tour to the first. For example, consider these two tours: a-b-c-d-e-f, and f-d-b-a-c-e. The PMX operator might choose at random the positions three and five, which would select the substring c-d-e from the first tour and b-a-c from the second tour. In the second tour, c is swapped with b (resulting in f-d-c-a-b-e), then d is swapped with a (resulting in f-a-c-d-b-e), and then e is swapped with b (resulting in f-a-c-d-e-b). The appropriate cities are also swapped in the first tour and the resulting tours are d-e-b-a-c-f and f-a-c-d-e-b. Notice that the substring c-d-e from the first tour has been imposed upon the second tour, regardless of how it affected the rest of the second tour. The same has happened to the first tour. Thus, by simply rearranging cities, both tours receive genetic information from the other. Refer to [3] for more details.

The mutation operator chose two random positions of the string and inverted all cities including and between those positions. This mutation operator is somewhat traumatic, especially if the graph of cities is asymmetrical, in which case the cost of going from city A to city B is different than the cost of going from B to A. In the current study the graph is symmetrical, so the cost between cities is independent of the order.

Elitism was also used in the GA. In every population, the GA discarded the worst performing individuals and copied the top performers into their place. Those top individuals are guaranteed to carry over unchanged to the next population, where they will either maintain their status as elites or be replaced by new ones.

Several experiments were performed on four different numbers of cities: ten, thirteen, thirty, and one hundred. Each experiment varied one of four different parameters of the GA, including the mutation rate, population size, maximum number of generations, and number of elites. The first experiment used default parameters that were chosen arbitrarily. Next, a pair of experiments was performed for each parameter, resulting in eight more experiments. The first of each pair increased the parameter slightly and the second increased the parameter by a large amount, all while keeping the other parameters constant. The GA ran fifty trials for each experiment and the best and average minimum tour lengths were used to compute accuracy. All nine of these experiments were performed for the 10-, 13-, 30-, and 100-city TSP. The city coordinates for the 10- and 13-city problems were generated randomly and the brute force algorithm was used to find their shortest tours. The city coordinates and shortest length for the 30-city problem came from [4]. The 100-city problem's coordinates and shortest length were obtained from [5]. All programs were implemented in Java and all tests were performed on an 800 MHz computer running Fedora Core 4 ®. The specific GA parameters used are shown in Table 1, in which the altered parameter is shown in italics. The average results of the experiments are shown in Table 2 and the best results are shown in Table 3. The tour lengths found are presented as a percentage of the optimal tour length, which is 100%. If a tour length is reported as 200% of optimal, it is twice as long as the optimal tour.

**Table 1. Specific GA Parameters Used for each Experiment**

| Exp. # | Mutation Rate | Pop Size | Maximum generations | Number of elites |
|---|---|---|---|---|
| 1 | 1% | 50 | 100 | 2 |
| 2 | *25%* | 50 | 100 | 2 |
| 3 | *75%* | 50 | 100 | 2 |
| 4 | 1% | *100* | 100 | 2 |
| 5 | 1% | *500* | 100 | 2 |
| 6 | 1% | 50 | *200* | 2 |
| 7 | 1% | 50 | *500* | 2 |
| 8 | 1% | 50 | 100 | *4* |
| 9 | 1% | 50 | 100 | *20* |

**Table 2. Average Percentage of Optimal for each Experiment**

| Exp # | 10 cities | 13 cities | 30 cities | 100 cities |
|---|---|---|---|---|
| 1 | 105.27% | 108.59% | 169.23% | 477.57% |
| 2 | 100.00% | 100.30% | 136.06% | 451.58% |
| 3 | 100.08% | 101.63% | 162.17% | 504.45% |
| 4 | 100.84% | 102.52% | 147.29% | 426.89% |
| 5 | 100.00% | 100.11% | 143.75% | 501.37% |
| 6 | 106.28% | 106.44% | 169.47% | 468.79% |
| 7 | 105.93% | 106.67% | 169.37% | 475.94% |
| 8 | 106.85% | 108.46% | 174.13% | 498.51% |
| 9 | 108.69% | 109.57% | 184.02% | 524.69% |

**Table 3. Best Percentage of Optimal for each Experiment**

| Exp # | 10 cities | 13 cities | 30 cities | 100 cities |
|---|---|---|---|---|
| 1 | 100.00% | 101.12% | 143.38% | 416.46% |
| 2 | 100.00% | 100.00% | 120.10% | 389.50% |
| 3 | 100.00% | 100.00% | 141.44% | 449.52% |
| 4 | 100.00% | 100.00% | 124.02% | 381.42% |
| 5 | 100.00% | 100.00% | 111.63% | 420.78% |
| 6 | 100.00% | 100.00% | 138.70% | 388.53% |
| 7 | 100.00% | 100.00% | 137.51% | 367.77% |
| 8 | 100.00% | 100.00% | 139.76% | 426.57% |
| 9 | 100.00% | 100.73% | 149.59% | 450.77% |

Table 4 presents the time it took each algorithm to find its answer. In Table 4, the brute force times for the 30- and 100-city problems are estimated from the time it took the brute force algorithm to solve the 13-city problem.

**Table 4. Time Comparison of Brute Force and GA**

| Number of cities | Brute force time | Average GA time |
|---|---|---|
| 10 | 87.02 seconds | 5.93 seconds |
| 13 | 2.13 days | 5.64 seconds |
| 30 | $2.49*10^{17}$ years | 10.11 seconds |
| 100 | $8.75*10^{145}$ years | 69.29 seconds |

The average percentages of optimal were used in the following analysis. While the best percentages are not analyzed in detail, similar conclusions would be drawn in a best-case analysis. As Table 2 shows, the GA performed well on both the 10- and 13-city TSP. The solutions were produced in a fraction of the time, as Table 4 shows. The GA did not perform as well on the 30- and 100-city problems. The decrease in accuracy was probably caused by the parameters of the GA. For the 30- and 100-city problems, the GA's accuracy improved over the default when the population size increased and in some cases when the maximum number of generations increased. These results suggest that greatly increasing the population size and allowing the GA to run longer would produce better results for larger numbers of cities.

In all cases, increasing the mutation rate to 25% (Experiment #2) greatly improved accuracy. For the 10-city problem, increasing the mutation rate produced perfect accuracy. Increasing the mutation rate to 75% (Experiment #3) produced worse accuracy than Experiment #2 in all cases, suggesting the mutation rate provides diminishing returns if it becomes too high.

In all the problems, increasing the population size to 100 (Experiment #4) produced better accuracy than the default size of 50. In the 10-, 13-, and 30-city problems, increasing the population size even more to 500 (Experiment #5) produced better accuracy than Experiment #4, suggesting that accuracy will increase as population size increases. A larger population allows more chances for the initial random population to create individuals with high fitness, which improves the search. In the 100-city problem, though, the population size of 500 did not outperform the population size of 100, suggesting that increasing population size alone is not a guaranteed way to increase accuracy. A larger population allows more chances of producing high-quality individuals from the start. However, even with a large population size there is a chance that the initial population will be filled with low-quality individuals because the initial population is randomly created.

In the 13- and 100-city problems, increasing the maximum number of generations to 200 and then 500 (Experiments #6 and #7, respectively) only slightly improved accuracy. In the 10- and 30-city problems, increasing the maximum number of generations slightly worsened the accuracy. These results suggest that increasing the maximum number of generations does not affect the outcome significantly. This conclusion is reasonable because the GA might always converge before the maximum number of generations is reached. In fact, in most of the experiments, the average number of generations used was much less than the maximum. However, in the 30- and 100-city problem, the number of generations required was frequently close to the maximum, suggesting that a larger number of generations might be more important for larger numbers of cities.

In almost all cases, increasing the number of elites decreased the GA's accuracy. These results make sense because keeping too many elites would crowd the population with the same individuals, being copied over and over again. The GA will converge much earlier, which increases the likelihood of converging on an answer that is further from optimal. The number of elites should be kept small.

One particularly interesting result came from increasing the mutation rate for the 30- and 100-city problem (Experiment #2). Though the average minimum tour length was still suboptimal, it was much closer than the length produced by the default GA values. In both the problems, the GA ran for the maximum number of generations – 100 – for every one of the fifty runs. This outcome and the success of the increased mutation rate for the 10- and 13-city problems suggests that Experiment #2 might

produce lengths for the 30- and 100-city problems that are closer to optimal if the GA is given more time (more generations).

Three more experiments were performed to test these conclusions. Experiment #10 increased both the mutation rate and the maximum number of generations. Experiment #11 increased the maximum number of generations even more, and Experiment #12 increased the population size as well. This data is summarized in Table 5, in which the parameters that are different from the default are italicized. The average results of the experiments are shown in Table 6 and the best results are shown in Table 7.

**Table 5. GA Parameters Used for Additional Experiments**

| Exp. # | Mutation Rate | Pop Size | Maximum generations | Number of elites |
|---|---|---|---|---|
| 10 | *25%* | 50 | *500* | 2 |
| 11 | *25%* | 50 | *5000* | 2 |
| 12 | *25%* | *100* | *500* | 2 |

**Table 6. Average Percentage of Optimal for each Experiment**

| Exp # | 30 cities | 100 cities |
|---|---|---|
| 10 | 104.07% | 267.61% |
| 11 | 101.17% | 113.85% |
| 12 | 106.09% | 316.70% |

**Table 7. Best Percentage of Optimal for each Experiment**

| Exp # | 30 cities | 100 cities |
|---|---|---|
| 10 | 100.00% | 228.74% |
| 11 | 100.00% | 107.35% |
| 12 | 100.00% | 285.14% |

Once again, only the average percentages of optimal were used in the analysis, but analysis of the best percentages would yield similar conclusions. In both the 30- and 100-city problems, increasing the mutation rate and maximum generations (Experiment #10) greatly improved accuracy. The GA produced the greatest accuracy when the maximum number of generations was increased even more (Experiment #11). Increasing the population size in Experiment #12 had a negative affect on the accuracy, affirming the conclusion that increasing the population size does not always increase accuracy. It appears the most important factor was increasing the maximum number of generations, i.e. the time the algorithm is allowed to run.

Allowing the GA to run until it converges (no restriction on maximum generations) can take a long time depending on the number of cities. For large numbers of cities, such a GA might have the same problem the brute force algorithm has: unreasonably long computation times. The maximum number of generations should be large enough so that the GA finds an accurate answer, but it should also be small enough to find an answer in a reasonable amount of time.

# 5. CONCLUSIONS

Some of the parameters of the GA must be finely tuned in order to produce a TSP solution that is close to optimal. Various crossover operators and mutation operators can also be used to obtain better results. Specialized crossover operators such as [6]'s heuristic crossover and [7]'s very greedy crossover have shown promising results for the TSP. Mutation operators could be made simpler or even more drastic than the one used in this study.

There are numerous possibilities for customizing any GA. Initial populations can be heuristically created instead of randomly, giving the GA a better place to start. A crossover operator can be made to produce either one or two children each time. There are many different ways the GA can rank individuals and select which ones to reproduce. [5] used a "man-machine" approach, in which the GA was customized to interact with human users in order to produce the best possible solution. Future work will examine further possibilities of customizing a GA for the TSP.

# 6. ACKNOWLEDGMENTS

Thanks to my student colleagues who proofread this paper for me many times.

# 7. REFERENCES

[1] Haupt, R. L. & Haupt, S. E. *Practical Genetic Algorithms* (2nd ed.). John Wiley & Sons, Hoboken, NJ, 2004.

[2] Russell, S. J. & Norvig, P. *Artificial Intelligence: A Modern Approach* (2nd ed.). Pearson Education, Upper Saddle River, NJ, 2003.

[3] Goldberg, D. E. & Lingle, R., Jr. Alleles, loci, and the traveling salesman problem. In J. J. Grefenstette (Ed.), *Proceedings of the first international conference on genetic algorithms and their applications (ICGA '85)* (Pittsburgh, PA, USA, July 24-26, '85). Lawrence Erlbaum Associates, Hillsdale, NJ, 1985, 154-159.

[4] Oliver, I. M., Smith, D. J., & Holland, J. R. C. (1987). A study of permutation crossover operators on the traveling salesman problem. In J. J. Grefenstette (Ed.), *Genetic algorithms and their applications: Proceedings of the second international conference on genetic algorithms (ICGA '87)* (Cambridge, MA, USA, July 28-31 '87). Lawrence Erlbaum Associates, Hillsdale, NJ, 1987, 224-230.

[5] Krolak, P., Felts, W., & Marble, G. A man-machine approach toward solving the traveling salesman problem [Electronic version]. *Communications of the ACM, 14,* (May 1971), 324-327.

[6] Grefenstette, J., Gopal, R., Rosmaita, B., & Van Gucht, D. Genetic algorithms for the traveling salesman problem. In J. J. Grefenstette (Ed.), *Proceedings of the first international conference on genetic algorithms and their applications (ICGA '85)* (Pittsburgh, PA, USA, July 24-26, '85). Lawrence Erlbaum Associates, Hillsdale, NJ, 1985, 160-168.

[7] Julstrom, B. A. Very greedy crossover in a genetic algorithm for the traveling salesman problem [Electronic version]. In *Symposium on applied computing: Proceedings of the ACM symposium on applied computing (SAC '95)* (Nashville, TN, USA, Feb. 26-28, '95). ACM Press, New York, NY, 324-328.