Genetic Algorithm Solution of the TSP Avoiding Special Crossover and Mutation

Article *in* Intelligent Automation and Soft Computing · January 2002

DOI: 10.1080/10798587.2000.10642829

Göktürk Üçoluk
Middle East Technical University
39 PUBLICATIONS **396** CITATIONS

Genetic Algorithm Solution of the TSP Avoiding Special Crossover and Mutation

Göktürk Üçoluk

Department of Computer Engineering
Middle East Technical University
06531 Ankara, Turkey

Email: ucoluk@ceng.metu.edu.tr

Abstract

Ordinary representations of permutations in Genetic Algorithms (GA) is handicapped with producing offspring which are not permutations at all. The conventional solution for crossover and mutation operations of permutations is to device 'special' operators. Unfortunately these operators suffer from violating the nature of crossover. Namely, considering the gene positions on the chromosome, these methods do not allow n-point crossover techniques which are known to favour building-block formations. In this work, an inversion sequence is proposed as the representation of a permutation. This sequence allows repetitive values and hence is robust under ordinary (n-point) crossover. There is a one-to-one mapping from ordinary permutation representation to the inversion sequence representation.

The proposed method is used for solving TSPs and is compared to the well known PMX special crossover method. It is observed that this method outperforms PMX in convergence rate by a factor which can be as high as 11.1 times, on a cost of obtaining slightly worse solutions on average.

Key Words: genetic algorithms, permutation representation, traveling salesman problem, crossover, partially mapped Crossover, PMX, TSP

1 Introduction

A well known computational problem is the Traveling Salesman Problem (TSP) which is known to be NP-complete. Here is the wording of it:

> *N points ('cities'), as well as the cost of traveling between every pair of them is given. Assume that a salesperson, starting from a given city, has to visit each city exactly once and hence make a round-trip. The aim is to find an optimal tour in which the total cost of the round-trip is minimized.*

More formally, the TSP can be formulated as a problem of graph theory: Given a graph G on a set of N vertices (cities), a closed sequence of edges in G (i.e. a cycle) which passes through each vertex of G exactly once is called a *Hamiltonian Cycle*. Given a complete weighted graph G on a set of N vertices the TSP is then the problem of finding the shortest Hamiltonian Cycle through G. From the computational point of view this means the determination of the particular permutation of the non-repeating sequence $1, 2, \ldots, N$ where the cities are numbered consecutively from 1 to N and the permutation represents the visiting order for which the weight sum is minimized.

The search space contains $N!$ permutations and since TSP is NP-complete, the corresponding optimization problem is NP-hard. The best known algorithms have exponential (deterministic) run time complexity. Such combinatorial optimization problems are in the domain of Genetic Algorithms interest. In the next section, the most popular method among conventional GA solutions of TSP will be reviewed. In section 3 an alternative solution will be introduced. A comparative experimental study will be covered in section 4 which is followed by the conclusion.

2 Conventional Approach

The first GA approach on TSP was by Brady [1] which was then followed by Grefenstette et al. [4]; Goldberg and Linge [6] and Oliver et al. For a detailed discussion on TSP a good reference is the study of Lawler et al. [11]. A perfect review article on GA for TSP is by Larranaga et al. [10].

In the conventional approach a chromosome which is devised to represent a solution constitutes of N (count of the cities) genes. Each gene holds a number which is a label of a city. So the n th gene holds the label of the city which is visited n th. In other words, the chromosome is a direct coding of a permutation of the sequence $1, 2, \ldots, N$.

The problem with this representation is obvious. Starting with a population of valid chromosomes, ordinary crossover and mutation operators cause problems. This is so, because offspring generated by means of the ordinary operators have a high chance of being invalid with some cities missing, and others repeated. A variety of methods that handle problems of this sort are introduced throughout the literature.

Solutions are observed to fall into one of the following three categories:

- **Disqualification:** The idea is to allow the generation of invalid chromosomes but assign such a low fitness values that they got eliminated in the forthcoming selection process. This simple method has a disadvantage of being time consuming. The genetic engine spends most of its time generating invalid chromosomes and then eliminating them.

- **Repairing:** In this approach invalid chromosomes are generated but then fed into an intermediate process where they are transformed into valid ones. Here the key idea is to do the least modification such that the merits of crossover are preserved. This type of methods are also time consuming.

- **Inventing Specialized Operators:** Instead of creating invalid chromosomes the GA operators are modified to generate only valid chromosomes.

The idea of disqualification proves itself to be extremely inefficient. Attempts for repairing can be found in Lidds study [12].

Falling into the third category and concerning permutation-respecting path crossover operators, the following operators are worth to mention:

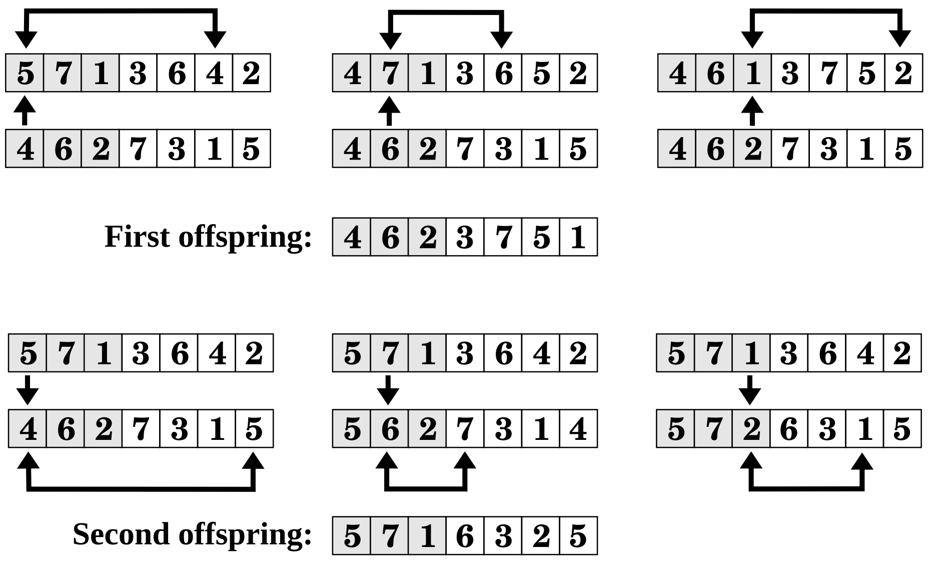
Partially - Mapped Crossover (PMX)	Goldberg and Lingle (1985) [6]
Order - Crossover (OX1)	Davis (1985) [2]
Order Based Crossover (OX2)	Syswerda (1991) [17]
Position Based Crossover (POS)	Syswerda (1991) [17]
Heuristic Crossover (HX)	Grefenstette (1987) [5]
Edge Recombination Crossover (ER)	Whitley et al. (1989) [18]
Sorted Match Crossover (SMX)	Brady (1985) [1]
Maximal Preservative Crossover (MPX)	Mühlenbein et al. (1988) [13]
Voting Recombination Crossover (VR)	Mühlenbein (1989) [14]
Alternating - Position Crossover (AP)	Larranaga et al. (1996) [9]

Among these PMX, ER and POS are quoted to be the fastest operators as far as the number of necessary iterations to reach convergence is concerned [10, 16]. The convergence rates of these three operators are observed to be similar. Literature also mentions that ER is observed to produce the best quality solutions [10, 18].

In this work we pick PMX among these and claim that our proposed crossover method produces slightly worse quality solution at a convergence rate which is faster then PMX by a factor of 3-5.

How does PMX function?

Given two parents s and t, PMX randomly picks a crossover point – like 1-point crossover. The child is then constructed in the following way. Starting with a copy of s, the positions between the crossover points are, one by one, set to the values of t in these positions. To keep the string a valid chromosome the cities in these positions are not just overwritten. To set position p to city c, the city in position p and city c swap positions. Below you see an example of this coding and special crossover technique for two sample permutations: $5, 7, 1, 3, 6, 4, 2$ and $4, 6, 2, 7, 3, 1, 5$



The resulting child has

1. between the crossover points, the same cities in the same positions as t, and
2. outside the crossover interval, the same cities in the same positions as s, where this is not in conflict with (1).

This idea can very easily be generalized to n-point crossover. Mutation is done by exchanging gene values in pairs (in a chromosome).

This method has the following draw backs:

1. The changes in the chromosomes are not confined to the exchanged portions. Therefore the *building-blocks* mechanism of EC [7] is damaged.
2. Mutations are not performed at single points.
3. Simple bit-string crossover and mutation implementations will not work.

In the following section a new technique for GA to deal with permutation encoding, where the representation is crossover and mutation robust, is introduced. This means that the offspring generated by crossover and mutation are still valid chromosomes and no special definitions for these operators are needed: the conventional bit-string crossover and mutation operators suffice.

3 Proposed Method

The proposed method is to describe a permutation by means of its inversions [8]. For a permutation $i_1, i_2, \dots, i_N$ of the set $\{1, 2, \dots, N\}$ we let a_j denote the number of integers in the permutation which precede j but are greater than j. So, a_j is a measure of how much out of order j is. The sequence of numbers $a_1, a_2, \dots, a_N$ is called the *inversion sequence* of the permutation $i_1, i_2, \dots, i_N$. For example the inversion sequence of the permutation $4, 6, 2, 7, 3, 1, 5$ is $5, 2, 3, 0, 2, 0, 0$. Here, for example, the 2 (which is the 5th element in the inversion sequence) is saying that there are exactly 2 elements in the permutation which are to the left of 5 and are greater than 5 (Yes this is true, they are: $6, 7$).

The inversion sequence $a_1, a_2, \dots, a_N$ satisfies the conditions

$$0 \leq a_i \leq N - i \qquad \text{for} \quad i = 1, 2, \dots, N$$

As seen there is no restriction on the elements which says $a_i = a_j$ is forbidden for $i \neq j$. This is of course very convenient for the crossover and mutation operations in GA.

Below two iterative algorithms are given. The first generates the inversion sequence of a given permutation and the second does the inverse (generates the corresponding permutation of a given inversion sequence).

Input *perm* : array holding the permutation
Output *inv* : array holding the inversion sequence

```
for $i \leftarrow 1..N$ do
    { $inv_i \leftarrow 0$
      $m \leftarrow 1$
      while $perm_m \neq i$ do
            { if $perm_m > i$ then $inv_i \leftarrow inv_i + 1$
              $m \leftarrow m + 1$ } }
```

Input *inv* : array holding the inversion sequence
Output *perm* : array holding the permutation
Uses *pos* : dummy array for intermediate result[1]

```
for $i \leftarrow N..1$ do
    { for $m \leftarrow i + 1..N$ do
            if $pos_m \geq inv_i + 1$ then $pos_m \leftarrow pos_m + 1$
      $pos_i \leftarrow inv_i + 1$ }
for $i \leftarrow 1..N$ do $perm_{pos_i} = i$
```

[1]The use of this array can be avoided by a more elaborated algorithm but this will not reduce the time complexity.

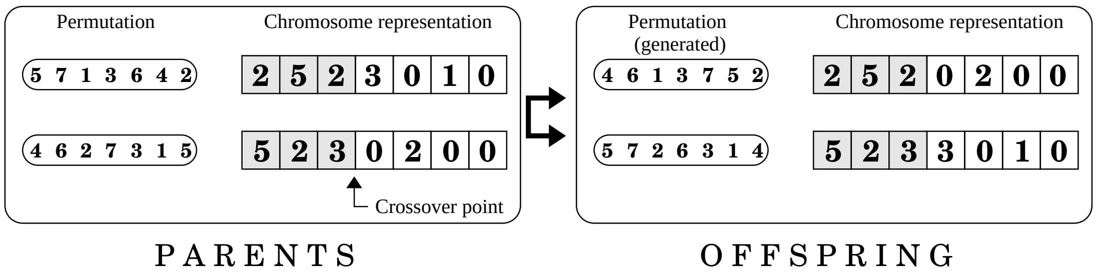
By this method a chromosome or a subsection of it, which has to keep a permutation, will consist of a sequence of N genes[2] where the allele of each element is a natural number. The maximal allele value allowed decreases by one at each element from the first position of the sequence to the last one.

In GA applications natural number valued genes are usually represented by bit strings which are the binary representation of that number. The limitation is very easily controlled by choosing a restricted bit length and/or a modulo operation. Except this limitation on the maximal values, which always is the case in GA applications with numerical alleles, there is no extra restriction or order that has to be preserved throughout the GA operations. Therefore, whatever crossover or mutation will produce will correspond to a valid permutation. Now there is a question to be answered:

What characteristics of the parents will be inherited by the offspring?

Assuming that an ordinary bit-string one-point crossover is performed on both components of the chromosome, we can state that the offspring will inherit characteristics from both parents. For one of the offspring, one of the parents, p_1, will provide the *displacement* information of some of the permutation elements (lets call them $\mathcal{E}'$) and the other parent, p_2 will provide a similar information for the remaining permutation elements ($\mathcal{E}''$). Of course the other offspring will receive the displacement information for $\mathcal{E}'$ and $\mathcal{E}''$ from p_2 and p_1, respectively. Similar properties can be stated for mutation.

Below is an example coding of the two permutations $5, 7, 1, 3, 6, 4, 2$ and $4, 6, 2, 7, 3, 1, 5$ which undergo an ordinary crossover that generates two offspring from them:



4 GA Solution of the TSP Problem

Both methods, PMX and the newly proposed one, are implemented as C programs. The differences are kept as local as possible. As a test bed a problem from the TSPLIB is chosen. This library is located at

http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/

provided and maintained by the Research Group on Discrete Optimization at Heidelberg University. As it is true for many problems in the TSPLIB pool, the problems used in our work have known optimal solutions. These values are quoted in our work. We refer to the given URL for further information and references.

The three test bed problems are symmetric TSPs and two of the data sets come from 'Real World problems':

[2]Actually $(N - 1)$ suffices, since inv_N is always zero.

bays29 : The road distances that connect 29 cities in Bavaria, Germany.

berlin52 : Street distances of the city of Berlin, Germany.

eil101 : An artificial 101 city problem created by Eilon and Christofides.

Both methods were run with the same settings of GA dynamics: 10-point crossover, pool size of 1000 chromosomes, 15% elitism, 0.007 mutation/gene_exchange. These values were set empirically to work well for both methods. Interestingly, while this tuning was performed, both PMX and the proposed method reacted coherently. That means, there was no special setting observed which would favour one of the methods.

The termination criteria was to continue to run the GA engine until there is no change in the fitness of the best of the pool for $max(200, Generation/3)$ generations. This figure, namely the count of generations in which no changes occurred, is subtracted from the total generation count. So, in our analysis the generation count of a GA run is taken to be the generation in which a change of the fitness (of the best) occurred for the last time.

To avoid the differences coming from the randomization of the initial pool both methods were run for 1000-5500 times and the comparison was made statistically.

Of course neither method converges always to the optimum but rather gets caught in local minima which are nearly optimal. Figures 1a, 2a, 3a displays the number distribution of the pool's bests of each of the runs for both the methods are given as histograms respectively for the test TSPs bays29, berlin52 and eil101. Similarly, figures 1b, 2b, 3b displays the number distribution of the count of iterations of each run for the same test TSPs.

Below you find tabulated parameters and results for the three test data sets.

Data set names	bays29	berlin52	eil101
Count of cities	29	52	101
Count of runs	5500	2000	1000
Optimal path length	2020	7542	629
Best path length found by PMX	2020	10007	10000
Best path length found by the proposed method	2026	10000	10000
Average path length & the standard deviation (in parenthesis) in path length, found by PMX	2239 (110)	9096 (447)	922 (148)
Average path length & the standard deviation in path length, found by by the proposed method	2261 (100)	10199 (539)	1069 (59)
Average iteration count & the standard deviation in iteration count, found by PMX	247 (48)	657 (229)	3457 (2405)
Average iteration count & the standard deviation in iteration count, found by the proposed method	112 (17)	201 (21)	311 (29)

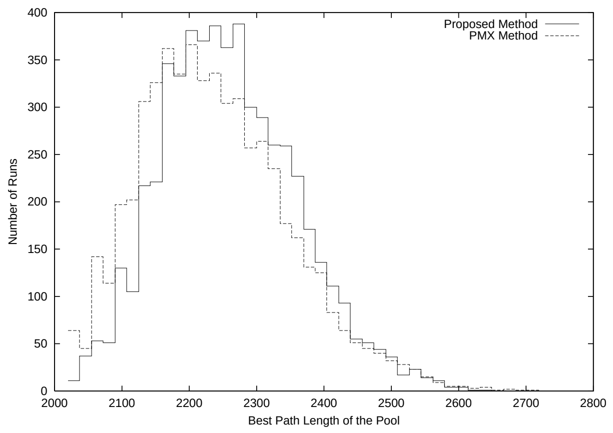


Figure 1a: Histogram of the pool's best over all runs of the bays29 problem.

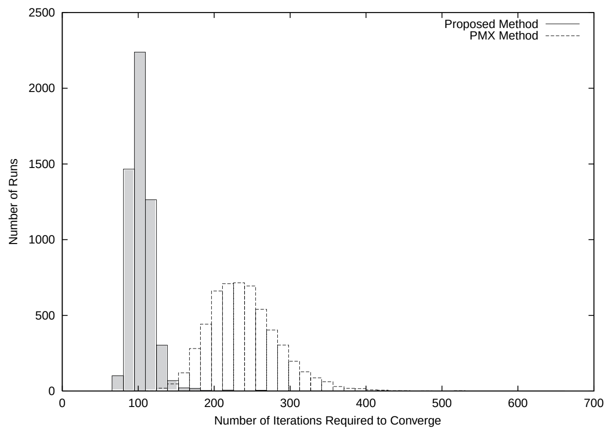


Figure 1b: Histogram of the iteration count over all runs of the bays29 problem.

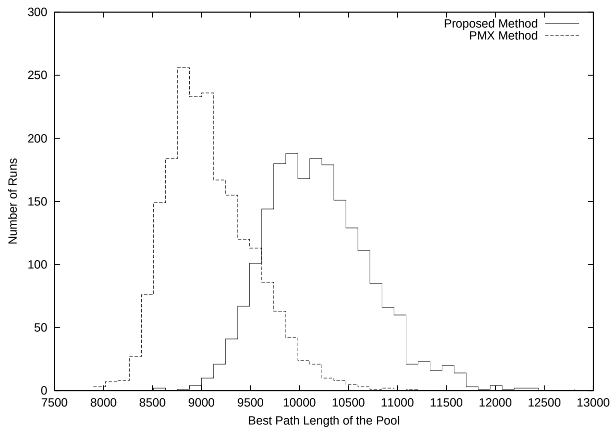


Figure 2a: Histogram of the pool's best over all runs of the berlin52 problem.

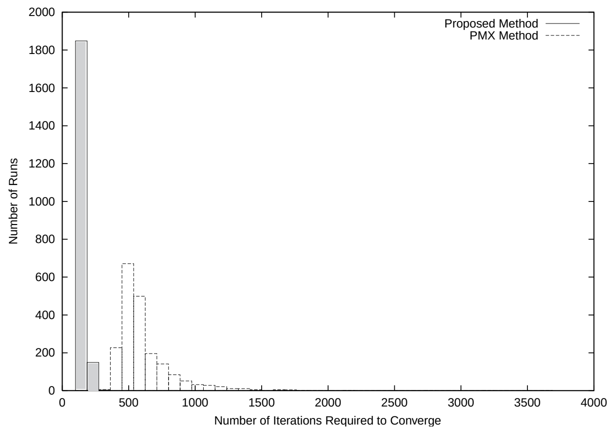


Figure 2b: Histogram of the iteration count over all runs of the berlin52 problem.

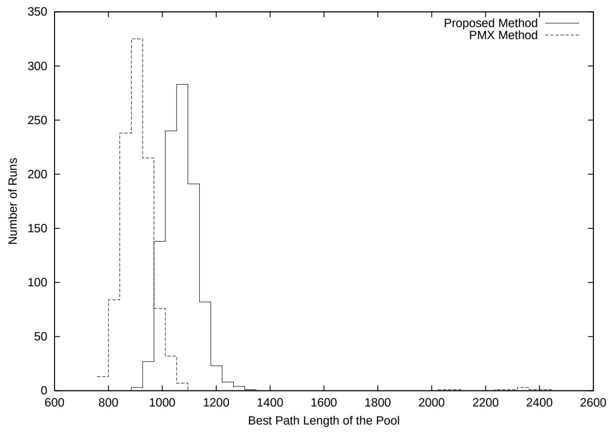


Figure 3a: Histogram of the pool's best over all runs of the eil101 problem.

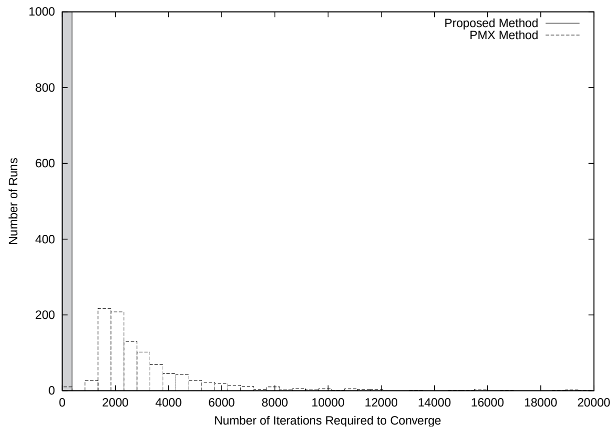


Figure 3b: Histogram of the iteration count over all runs of the eil101 problem.

Observing the results we conclude that:

- Regarding the best solution found for each TSP there is no observed systematic difference between PMX and the proposed method.
- On the average PMX is producing 1%-15% better solutions than the proposed method. The standard deviation of this value varies for both method and is not conclusive.
- The proposed method outperforms PMX by factors that range from 2.2 times to 11.1 times when the converge rate is concerned. The proposed method converges much faster than PMX does. Also standard deviation figures of these values shows that the proposed method is much more stable in the iteration count needed to converge.

5 Conclusion

A new method for representing permutations as GA chromosomes has been introduced. In contrast to the conventional ones this proposed representation is not handicapped under crossover and mutation. The proposed method was used in a TSP test bed and has proven itself to be almost as good as the conventional method as far as the solution quality (i.e. finding the optimum solution) is concerned. The comparative study of the results shows that the new method outperforms the conventional PMX method by a factor which can be as high as 11.1 in convergence rate.

Other discrete optimization problems, like scheduling or timetabling problems that involve permutation representations and are attempted to be solved by means of GA approaches, may also benefit from the proposed method.

References

[1] Brady, R.M. "Optimization Strategies Gleaned from Biological Evolution." *Nature* 317, 1985, pp. 804.

[2] Davis, L. "Applying Adaptive Algorithms to Epistatic Domains." *Proceedings of the International Joint Conference on Artificial Intelligence*, 1985, pp. 162-164.

[3] Even, S. *Algorithmic Combinatorics*. The Macmillan Company, NY, 1973.

[4] Grefenstette, J., R. Gopal, B. Rosmaita, and D. Van Gucht. "Genetic Algorithms for the TSP." *Proceedings of the First International Conference on Genetic Algorithms and Their Applications*, edited by Grefenstette J., Lawrence Erlbaum, Hillsdale, New Jersey, 1985, pp. 160-165.

[5] Grefenstette, J. "Incorporating Problem Specific Knowledge into Genetic Algorithms." *Genetic Algorithms and Simulated Annealing*, edited by Davis L., Morgan Kaufmann, Los Altos, CA, pp. 42-60, 1987.

[6] Goldberg, D.E., and R. Lingle. "Alleles, Loci, and the Traveling Salesman Problem." *Proceedings of the First International Conference on Genetic Algorithms and Their Application*, edited by Grefenstette J., Lawrence Erlbaum Associates, Hillsdale, NJ, 1985, pp. 154-159.

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

[8] Hall, M. *American Mathematical Society, Proceedings of the Symposium on Pure Mathematics*, 6, 1963, pp. 203.

[9] Larranaga, P., C.M.H. Kuijpers, M. Poza, and R.H. Murga. "Decomposing Bayesian Networks: Triangulation of the Moral Graph with Genetic Algorithms." *Statistics and Computing*, 1996.

[10] Larranaga, P., C. Kuijpers, R. Murga, I. Inza, and S. Dizdarevic. "Genetic Algorithms for the Traveling Salesman Problem: A Review of Representations and Operators." *Artificial Intelligence Review*, 13, 1999, pp. 129-170.

[11] Lawler, E.L., J.K. Lenstra, A.H.G. Rinnooy Kan, and D.B. Shmoys (Eds.). *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*. Wiley, Chichester, 1985.

[12] Lidd, M.L. *The Traveling Salesman Problem Domain Application of a Fundamentally New Approach to Utilizing Genetic Algorithms*. Technical Report, MITRE Corporation, 1991.

[13] Mühlenbein, H., M. Gorges-Schleuter, and O. Kramer. "Evolution Algorithms in Combinatorial Optimization." *Parallel Computing*, 7, 1988, pp. 65-85.

[14] Mühlenbein, H. "Parallel Genetic Algorithms, Population Genetics and Combinatorial Optimization." *Proceedings on the Third International Conference on Genetic Algorithms*, edited by Schaffer J., Morgan Kaufmann Publishers, Los Altos, CA, 1989, pp. 416-421.

[15] Oliver, M., D.J. Smith, and J.R.C. Holland. "A Study of Permutation Crossover Operators on the Traveling Salesman Problem." *Proceedings of the Second International Conference on Genetic Algorithms*, edited by Grefenstette J., Lawrence Erlbaum Associates, Hillsdale, NJ, 1987, pp. 224-230.

[16] Starkweather, T., S. McDaniel, K. Mathias, C. Whitley, and D. Whitley. "A Comparison of Genetic Sequencing Operators." *Proceedings on the Fourth International Conference on Genetic Algorithms*, edited by Belew R. and Booker L., Morgan Kaufmann Publishers, Los Altos, CA, 1991, pp. 69-76.

[17] Syswerda, G. "Schedule Optimization Using Genetic Algorithms." *Handbook of Genetic Algorithms*, edited by Davis L., Van Nostrand Reinhold, New York, 1991, pp. 332-349.

[18] Whitley, D., T. Starkweather, and D. Shaner. "The Traveling Salesman and Sequence Scheduling: Quality Solutions Using Genetic Edge Recombination." *Handbook of Genetic Algorithms*, edited by Davis L., Van Nostrand Reinhold, New York, 1991, pp. 350-372.