A Genetic Algorithm with New Local Operators for Multiple Traveling Salesman Problems

Kin-Ming Lo 1, Wei-Ying Yi 1 Pak-Kan Wong 1 Kwong-Sak Leung 1 Yee Leung 2 Sui-Tung Mak 3

1 Department of Computer Science and Engineering, The Chinese University of Hong Kong
Hong Kong, China
E-mail: kmlo@cse.cuhk.edu.hk

2 Institute Of Future Cities, The Chinese University of Hong Kong
Hong Kong, China

3 Department of Electronics and Computer Science, University of Southampton
University Road, Southampton SO17 1BJ, United Kingdom

Received 20 June 2017
Accepted 21 January 2018

Abstract

Multiple Traveling Salesman Problem (MTSP) is able to model and solve various real-life applications such as multiple scheduling, multiple vehicle routing and multiple path planning problems, etc. While Traveling Salesman Problem (TSP) focuses on searching a path of minimum traveling distance to visit all cities exactly once by one salesman, the objective of the MTSP is to find m paths for m salesman with a minimized total cost - the sum of traveling distances of all salesmen through all of the respective cities covered. They have to start from a designated depot which is the departing and returning location of all salesmen. Since the MTSP is a NP-hard problem, a new effective Genetic Algorithm with Local operators (GAL) is proposed in this paper to solve the MTSP and generate high quality solution within a reasonable amount of time for real-life applications. Two new local operators, Branch and Bound (BaB) and Cross Elimination (CE), are designed to speed up the convergence of the search process and improve the solution quality. Results demonstrate that GAL finds a better set of paths with a 9.62% saving on average in cost comparing to two existing MTSP algorithms.

Keywords: Multiple Traveling Salesman Problem, Genetic Algorithm, Branch and Bound algorithm, Local operators

1. Introduction

Traveling Salesman Problem (TSP) is a widely studied optimization problem. Given a set of cities and the distances between them, the TSP is defined as a path searching problem for a salesman to visit all cities exactly once, starting and ending at the same city. The objective of a TSP is to minimize the traveling distance of the salesman. Many real-life applications can be modeled as a TSP, such as the printed-circuit-boards drilling problem 1, the order-picking problem 2 and the wallpaper minimization problem 3.

Multiple Traveling salesman Problem (MTSP) is an extension of the TSP, which has more than one salesman deployed concurrently to visit the cities. All salesmen depart from and return to the same depot. Except for the depot, each of the cities can only

* Corresponding author
be visited by exactly one salesman. Minimizing the total cost of all paths taken by the salesmen is the goal of an MTSP. Some typical applications of the MTSP are path planning, scheduling and school-bus routing.

Our work is motivated by the multiple quadcopter paths planning problem. Compared to fixed-wing aerial vehicles, quadcopters have different aerodynamics. As the flight time is highly limited by the internal batteries, an effective path planning algorithm can help to accomplish as many tasks as possible within a deployment of multiple quadcopters. Such a problem can be modeled as a three-dimensional MTSP.

The MTSP can also be extended with different constraints or objectives, such as the multiple objective, multiple depot, multiple vehicle routing problem, traveling salesman problem with multiple Stacks etc. Designing efficient algorithms for the MTSP can also benefit these problems. Currently, the existing works mostly focus on modeling real-life scenarios into MTSP but only a few have the potential to solve the MTSP efficiently.

Because the MTSP is a NP-hard problem, there is no known polynomial-time algorithm for the MTSP. Therefore, it is more realistic to resolve the problem by heuristic algorithms. In this paper, an efficient algorithm based on genetic algorithm (GA), called Genetic Algorithm with Local operators (GAL) is proposed and developed to solve the MTSP. The major contributions of this paper are: 1) we have designed a GA-based algorithm for solving MTSPs effectively and; 2) we have proposed two novel local operators for the GA to improve its convergence speed.

We organize the paper as follows. Section 2 reviews the related research. In Section 3, the problem definition and formulation are discussed. The algorithm is described in detail in Section 4. Experimental results and discussions are reported in Section 5. The conclusion is presented in Section 6.

2. Related Work

The practical algorithms to solve the MTSP can be categorized into exact and meta-heuristics algorithms. The selection of an algorithm mainly depends on the problem size. Exact algorithms are suitable for small-scale problems to get the optimal solution. Due to the NP-hard nature of the MTSP, heuristic algorithms are more popular in solving large-scale MTSPs. The cutting-planes algorithm was used to optimally solve the problem with less than 100 cities.

An extended simulated annealing algorithm was developed to solve augmented TSPs and MTSPs. The proposed algorithm does not require transformation from an MTSP form to the standard TSP form before solving the problem.

A modified Genetic Algorithm (GA) with a new crossover framework, which balances the computation time and the quality of the solution, was introduced in the paper to model the hot rolling problem into the MTSP and handle by GA.

A two-part chromosome encoding scheme for GA was proposed. By eliminating redundant solutions, the size of solution space reduces and convergence is improved. The experiments indicated that this encoding method produces the best solution, comparing to the existing encoding methods with the same execution time. For this reason, subsequent works on solving MTSPs using GA mostly adopted this chromosome encoding method in their work.

GA with 2-opt local operator for solving the MTSP was invented. In the paper, the initial population is generated using the nearest neighbor method to introduce greedy initial population to enhance the convergence speed in the early stages. 2-opt local operator was adopted to avoid pre-mature convergence of the algorithm. Various mutation and crossover operators for MTSPs were designed and compared by the authors. The result showed proper operator design can boost the performance.

When the number of cities in an MTSP is relatively large (>500 cities), Ant Colony Optimization (ACO) was likely to produce better solutions comparing to GA on MTSPs. The result had been further improved by using the elite ACO. The initial paths are generated by the Sweep Algorithm to cluster the cities by the polar coordinate angles between the cities and the depot, which creates a
greedy initial population. A 3-opt local operator is also applied to speed up the convergence. To enhance the capability to escape from local optimum points, the same authors designed an insert, swap, and 2 opt algorithm. Five local search schemes were introduced by the authors to improve the result without much increased time complexity.

Inspired by the existing works mentioned above, the work done in this paper further improves the convergence speed of using GA to solve the MTSP with two novel local operators.

3. Methodology

3.1. Problem Definition and Formulation

The MTSP is defined as follows: given \( m \) salesmen, \( n \) cities and the visiting cost \( C \) (an \( n \) by \( n \) matrix) between the cities, search the visiting sequences for the \( m \) salesmen such that the total visiting cost of all salesmen is minimized. Each city can only be visited exactly once by one of the salesmen. All salesmen start from and return to the same depot (which is not counted as a city to be visited). Each salesman has to visit at least 1 city and at most \( P \) cities.

The assignment-based integer programming representation for the MTSP is presented as follows: The cities and the visiting routes can be represented as an undirected and complete graph \( G = (V, E) \) which contains a set \( V \) of \( n \) vertices (the cities) and a set \( E \) of \( n^2 \) (the routes between cities) edges, where \( V_0 \in V \) is denoted as the depot for the salesmen. The rest of the \( n - 1 \) vertices \( V \setminus \{V_0\} \) represent the cities to be visited. Each vertex \( V_i \) is associated with a position \((x_i, y_i)\) in the Cartesian-coordinate system. \( E_{ij} \in E \) is the edge traveling from vertex \( V_i \) to \( V_j \) associated with a traveling cost \( C_{ij} \), which is the Euclidean distance between the vertices \( V_i \) and \( V_j \). Therefore, the visiting cost is symmetric, i.e. \( C_{ij} = C_{ji} \).

A binary variable \( X_{ij} \) is defined to indicate the usage of an edge from vertex \( V_i \) to \( V_j \) in the solution. It equals to 1 if edge \( E_{ij} \) is included in the solution path and 0 otherwise. \( P \) is the maximum number of cities that one salesman can visit in a path. Therefore, the problem is formulated as

\[
\begin{align*}
\text{minimize} & \quad \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} C_{ij}X_{ij} \\
\text{subject to} & \quad x_{ij} \in \{0, 1\}, \forall e_i, e_j \in E \\
& \quad \sum_{i=1}^{n-1} X_{0j} = m \\
& \quad \sum_{i=1}^{n-1} X_{i0} = m \\
& \quad \sum_{i=0}^{n-1} X_{ij} = 1, j = 1...n - 1 \\
& \quad \sum_{j=0}^{n-1} X_{ij} = 1, i = 1...n - 1 \\
& \quad u_i - u_j + P \cdot X_{ij} \leq P - 1, for 1 \leq i \neq j \leq n - 1
\end{align*}
\]

Constraints (2) and (3) ensure that there are exactly \( m \) salesmen departing from and returning to the depot. Each vertex is entered and exited by exactly one salesman, as ensured by constraints (4) and (5). Constraint (6) is called the subtour elimination constraints, which prevents the paths from being formed without depot, where \( u_i \) represents the number of cities visited from the depot up to vertex \( i \) for any salesman.

3.2. Genetic Algorithm Solution

GA is a multipoint stochastic search algorithm which was first proposed by Holland. It simulates the natural evolution process to generate solutions and search the optimal solution(s) for optimization problems. Considerable amount of researches on using GA to solve TSP or MTSP have been done successfully with satisfactory results.

In GA, the candidate solution is usually encoded in a numeric vector called 'chromosome'. Each number inside the chromosome is considered as a gene. Then a set of individuals containing the randomly generated chromosomes forms a population.
Each individual is evolved by selection, mutation and crossover operators. The mutation operator randomly changes one or more genes to keep the diversity of the population, where the crossover operator attempts to combine the advantages of the parent chromosomes to form the child chromosome. A fitness value is computed to evaluate the quality of a chromosome. For the MTSP, the fitness value equals to the total traveling costs of all salesmen. Hence, the optimal solution is found when the fitness value is minimized.

Our proposed algorithm is also a GA. The complete process of the proposed algorithm is shown in Figure 1. The termination criteria can be set as fixed number of generations, fixed execution time or termination time if no significant improvement can be made compared to the preceding generations.

3.2.1. Chromosome Representation

There are several ways to encode an MTSP solution into a chromosome, including one chromosome, two chromosomes, two-part chromosome etc. With the condensed solution space, experimental results have demonstrated that two-part chromosome performs the best in terms of convergence speed and quality of solution. Hence, two-part chromosome representation is adopted in this work.

Each chromosome is composed of a numeric vector with length \( n + m - 1 \). The first \( n - 1 \) genes represent the visiting permutation for the salesmen. The remaining \( m \) genes represent the number of cities for each salesman to visit. One of the chromosome representation examples is shown as follows, where \( G_i \) represents each city visited and \( L_j \) is the number of cities visited by salesman \( j \).

\[
\text{Chromosome} = [G_1, G_2, \ldots, G_{n-2}, G_{n-1}, L_1, L_2, \ldots, L_{m-1}, L_m]
\]

Because all salesmen must depart from and return to the depot (\( V_0 \)), this city is not included inside the chromosome to save the memory usage. A chromosome representation example with 11 cities and 3 salesmen is illustrated in Figure 2. In this example, the first salesman is responsible for visiting 5 cities as stated in the second part of the chromosome. The visiting cities’ permutation of the first salesman is 0 (Depot) → 9 → 8 → 1 → 3 → 6 → 0 (Depot). The visiting cities’ permutation of the second salesman is 0 (Depot) → 7 → 2 → 10 → 0 (Depot). Finally, the visited cities’ permutation of the third salesman is 0 (Depot) → 4 → 5 → 0 (Depot).

3.2.2. Initial Population Generation

A better quality of initial population can improve the search efficiency since random permutation of cities is unlikely to produce informative routes in the evolution. A method modified from the Sweep Line algorithm is adopted in our work. The algorithm generates a sweep line (dotted line in Figure 3) from the depot to an arbitrary city. By sweeping the line clockwise with the depot as center, a path ordered by the polar angle is built as shown in Figure 3. This approach assumes that if the cities have similar polar coordinate angles, they tend to be closer to each other.

Half of the initial population contain the generated paths using the Sweep Line algorithm, while the
remaining paths are randomly generated as random permutations to maintain the diversity in the population. After that, for each chromosome produced by the Sweep Line algorithm, a segment of genes (i.e. 1% genes of the chromosome) is selected randomly, the genes inside the segment are re-ordered using the nearest neighbour method as follows: For the genes in the selected segment, one of the genes is chosen randomly as the starting gene. For the next gene, the gene closest to the previous gene in the selected segment will be chosen. When all the genes in the newly constructed segment are re-ordered, they replace the original segment in the chromosome.

![Fig. 3. An initial path generated by the Sweep line algorithm](image)

Next, the number of cities for each salesman encoded in the second part of the chromosome are initially evenly distributed. It avoids the paths having excessive lengths at the beginning. The size of the initial population is much larger than the population size of later generations, and only 1.5% (50 over 3000) of the initial chromosomes are selected to be evolved based on their diversity and fitness value.

### 3.2.3 Genetic Operators

Genetic operators are essential for GA to evolve the chromosomes. They significantly influence the search ability and convergence speed. Proper design of the genetic operators can further prevent pre-mature convergence, which means the solution is stuck in a local optimum. Hence, the choice of proper genetic operators is essential.

As aforementioned, there are constraints on the final solution. The cities inside the first part of the chromosome must appear only once to ensure that each city is visited only once. Besides, the sum of the values in the genes of the second part of the chromosome must be equal to the total number of cities to be visited. Also, there is a maximum number of cities for each salesman to visit. Due to these constraints, canonical genetic operators cannot be directly adopted in this chromosome encoding scheme. Therefore, modified genetic operators are used to evolve the chromosomes.

Since the first and second parts of the chromosome contain different information and should not be mixed, separate genetic operators for each part are required.

There are two mutation operators for the first part of the chromosome, which are the **random swap** and the **reverse swap** operators. For the random swap operator, two random distinct positions \((G_i \text{ and } G_j)\) are selected, where \(i \neq j\), the genes in these two positions are exchanged. For the reverse swap operator, two random distinct positions are selected to define segment, the position of the genes inside the segment are reversed. Examples are shown in Figures 4 and 5 respectively.

![Fig. 4. Random swap operator](image)

For the second part of the chromosome, which controls the number of cities for each salesman to visit, **random distribution mutation** is applied as mutation operator. Two random distinct positions \((i \text{ and } j, \text{ where } i \neq j)\) from the second part of the chromosome are selected. The gene in \(G_i\) will be incremented by one, and the gene in \(G_j\) will be decremented by one as shown in Figure 6. Then we have to check whether the result chromosome still fulfills the number of cities per salesmen constraint, which means \(G_i \geq 1\) and \(G_j \leq P\), where \(P\) is the maximum number of cities allowed to visit for each salesman.

In the example shown in Figure 6, originally, salesman 2 has to visit 7 cities while salesman 4 has to visit 9 cities. After mutation, salesman 2 now visits 8 cities while salesman 4 visits 8 cities.

![Fig. 5. Reverse swap operator](image)
Edge recombination crossover operator is modified to crossover the first part of the chromosome. This operator has been validated to be efficient in solving the TSP. The key idea of this operator is that if an edge appears in both parent chromosomes, it is believed to be a good edge and has higher chance to be present in the optimal solution. The neighboring information is stored in an edge table. The table size is \( n \times 4 \), where each row represents a vertex, and the columns contain the neighboring vertices information of that vertex. As each vertex appears once in both parent chromosomes, and it contains at most two neighbours in each parent chromosome, hence the column size is determined to be 4. The detail of this operator is presented in Algorithm 1.

**Algorithm 1: Edge recombination crossover operator**

- **Data**: Two parent chromosomes \( P_1, P_2 \)
- **Result**: Two offspring chromosomes \( C_1, C_2 \)

1. Record the neighbour vertices of both \( P_1 \) and \( P_2 \) in the edge table;
2. **for Each row in the edge table do**
   - **if repeated vertices are found in that row then**
     - This vertex has appeared in both parents, mark it;
   - **end**
3. Copy the first 1/3 genes from \( P_1 \) to \( C_1 \);
4. Put the remaining vertices into an available vertex list;
5. **while \( C_1 \) is not completed do**
6. Use the last gene of \( C_1 \) as index, look up that row in edge table;
7. **if marked vertex(es) is found and at least one is in available vertex list then**
8. Select it as next gene. If there exists more than one marked vertex, pick the next gene randomly among them;
9. **else**
10. Look up the nearest 5 neighbour locations using last gene of \( C_1 \) as index;
11. **if at least one is in available vertex list then**
12. Select the closest available neighbour as next gene
13. **else**
14. Randomly pick a location from available vertex list
15. Remove the next gene vertex from available vertex list;
16. Put the selected next gene behind the last gene of \( C_1 \);
17. **end**
18. Repeat Step 6 to Step 23 using \( P_2 \) to replace \( P_1 \) to generate \( C_2 \);
19. **return** \( C_1 \) and \( C_2 \);

3.2.4. Local Operators

In the general GA evolution process, the convergence power is determined by the mutation and crossover operators. Local operators are used in this work to boost the convergence rate. The operators use local search techniques to introduce greedy chromosome segments into the population. To prevent premature convergence, the operators should only be executed in between specific generations (e.g. 100 generations) and inside a small portion (e.g. to the top 5 individuals only) of the population. We have designed two local operators to optimize the paths at different scales. The Cross Elimination operator (CE) works in a global manner, it can reduce the total traveling cost by rearranging the visiting order of a large number of cities. In addition, the Branch-and-Bound operator (BaB) targets...
at producing an optimal path for a small segment of the cities from one salesman path.

**Cross Elimination Operator (CE)** This local operator actively searches for the crosses formed inside the paths, and attempts to remove them by reordering the cities sequence. The fundamental concept of CE operator is similar to the 2-opt operator \(^{27}\). If a cross is formed inside a path (the sequence of cities visited by one salesman), it means the path is not optimal. The cross searching technique of CE is based on the Bentley Ottmann algorithm \(^{28}\). It is a sweep-line algorithm for detecting the crosses of a set of line segments. The crosses in the MTSP can be classified into two types, namely the intra salesman path crosses and the inter salesmen path crosses. An intra salesman path cross is formed by the edges in a single salesman path, while an inter salesmen path cross is formed by the edges across two different salesmen paths.

The CE operator needs to be performed carefully to avoid incorrect result from being produced. It is because solving a cross may generate new cross(es) or remove other existing cross(es). An example is shown in Figures 7 and 8. Therefore, before solving a cross, it should be checked to see whether the cross still exists in the path or not. All the newly generated crosses will be handled in the next cycle of the process. To determine the cross solving order, a list containing the estimated cost saving of solving each cross is created and sorted by descending order. The elimination will begin from the cross which maximizes the cost saving first.

Solving the intra salesman path crosses is always feasible as the number of cities visited by each salesman remains unchanged and total traveling cost always decrease. For the inter salesman path cross, the solving process is more complicated. There exists two different options to handle it, as shown in Figure 9 to 11. It is possible that the new solution violates the path length constraint (i.e., each salesman has to visit at least 1 city and at most \(P\) cities). We can see that one of the newly generated paths becomes longer while the other path becomes shorter in Figure 11, that means one salesman will visit 5 cities and the other salesman will visit 3 cities. Hence, when handling inter salesman path crosses, we need to consider whether the result can still satisfy the path length constraints or not and which way to solve the cross such that most costs are saved. This can be achieved by estimating the number of cities and the fitness of the paths after cross solving respectively.
After a solving sequence is built, crosses solving can be executed in order. As stated in the previous paragraph, the crosses in the path may change after a cross is solved, some new crosses may be formed and are not detected by the initial checking. The cross solving process will iterate for several cycles (at most 5 cycles) or stop when no major improvement (at least 1% improvement in fitness) can be achieved.

Branch-and-Bound Operator (BaB) This local operator is based on the Branch-and-Bound algorithm. The algorithm is designed for searching the optimal solution of discrete and combinatorial optimization problems. It consists of a systematic enumeration of candidate solutions by the means of state-space search: the set of candidate solutions form a rooted tree with the full set at the root. The algorithm explores branches of this tree, which represent subsets of the solution set. Before enumerating the candidate solutions of a branch, the branch is checked against the upper and/or lower estimated bounds on the optimal solution, and is discarded if it cannot produce a better solution than the best one found so far by the algorithm.

Previous works have applied Branch-and-Bound to calculate the optimal solution of a TSP. Since the algorithm has high time complexity \(O(n^3)\) in average, it requires an excessive amount of time to find the solution when the number of cities is large. In this local operator, the Branch-and-Bound algorithm is only applied on a subset of a single salesman path. The subset is randomly selected from a continuous segment of a single salesman path. This reduces the running time of the operator, and introduces locally optimal path to the population. An example is shown in Figure 12. The BaB operator is applied to the cities marked with solid circles. The vertices between those cities are reordered by Branch-and-Bound to achieve a locally optimal TSP path. From our testing, we randomly select a total of 10% genes to be processed by the BaB piece by piece. Each piece has at most 5 cities to balance the computational time and performance.

4. Evaluation

4.1. Experimental Setting

In this section, we will evaluate the performance of GAL. Firstly, the experiments focus on measuring the convergence power of the new local operators. Secondly, we compare GAL with the existing MTSP solving approaches with respect to computational time and solution quality.

GAL is coded in Java and compiled with JDK 1.8. Experiments are run using one single thread on a PC with Intel Core i7-3370 CPU @ 3.4 GHz and 32 GB RAM. Each test case is run for 20 times to reduce random fluctuations in evaluation. A problem set from the TSPLIB is used as benchmark in the experiments. The problem set contains 6 problems with the numbers of cities ranging from 76 to 1002. In this benchmark, there is a maximum cities constraint, which sets an upper limit on the number of cities each salesman can visit. The benchmark details are listed in Table 1. Selected parameters for GAL are listed in Table 2.

<table>
<thead>
<tr>
<th>Instance</th>
<th>No of cities</th>
<th>Number of salesmen</th>
<th>Max cities</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pr76</td>
<td>76</td>
<td></td>
<td>20</td>
</tr>
<tr>
<td>Pr152</td>
<td>152</td>
<td></td>
<td>40</td>
</tr>
<tr>
<td>Pr226</td>
<td>226</td>
<td></td>
<td>50</td>
</tr>
<tr>
<td>Pr299</td>
<td>299</td>
<td>5/10/15</td>
<td>70</td>
</tr>
<tr>
<td>Pr439</td>
<td>439</td>
<td></td>
<td>100</td>
</tr>
<tr>
<td>Pr1002</td>
<td>1002</td>
<td></td>
<td>220</td>
</tr>
</tbody>
</table>

Fig. 12. An example to applying BaB operator to a single salesman path.
### Table 2. Parameters for GAL

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial Population Size</td>
<td>3000</td>
</tr>
<tr>
<td>Population Size</td>
<td>50</td>
</tr>
<tr>
<td>Random Swap Rate</td>
<td>30%</td>
</tr>
<tr>
<td>Reverse Swap Rate</td>
<td>10%</td>
</tr>
<tr>
<td>Crossover Rate</td>
<td>40%</td>
</tr>
<tr>
<td>Random Distribution Rate</td>
<td>20%</td>
</tr>
<tr>
<td>Local Operator</td>
<td>Every 100 generations to the top 1,2,4 individuals CE ran before BaB</td>
</tr>
</tbody>
</table>

4.2. **Comparing Different Local Operators under different numbers of salesmen**

In this paper, two novel local operators - BaB and CE are formulated to speed up the convergence speed of the proposed algorithm. This experiment aims at comparing the convergence power of the operators under different numbers of salesmen. Hence, we set the maximum number of generations to a fixed value, which is used as the termination criterion. The results are reported in Table 3. Combining both local operators (BaB and CE) in GAL obtains the best-fitness solution compared to other settings in most of the test cases. The fitness is defined as the sum of the salesmen’s path costs, which is the smaller the better.

The convergence curves with 5 agents are shown in Figures 13 and 14. With a larger problem size, the solutions converge faster when both operators are used. The average fitness values are improved by 4.1%, 28.1%, 36.7% with 5, 10, 15 salesmen respectively compared to the results without local operators.

From the experiment, we observed BaB works poorly with a small ratio of number of cities / number of salesmen, which means for each salesman visiting path, it is more likely to be a short path. We believe such kind of path may achieve local optimum in early GA stage. Hence, applying BaB will not help, since MTSP global optimization requires inter-salesmen cities exchange.

To boost GAL, a threshold for BaB can be introduced. When the fitness improvement obtained by BaB is lower than a threshold (k %), it will be disabled for certain cycles. It will help to save time and avoid creating disarrangement for finding better solution by cities exchange.

The rates of local operators can also be adjusted to fit different problems. If the application is time critical, BaB can be disabled to obtain the acceptable results in the quickest way. If the computational resource and time are adequate, the application is cost critical, using both operators will bring the best benefit to the outcome.

![Fig. 13. Convergence diagram with 152 cities with 5 salesmen](image1)

![Fig. 14. Convergence diagram with 1002 cities with 5 salesmen](image2)
Table 3: Fitness comparison with different local operators and number of salesmen

<table>
<thead>
<tr>
<th>Problem</th>
<th>Local Operator</th>
<th>Best</th>
<th>Avg</th>
<th>Worse</th>
<th>Time</th>
<th>Best</th>
<th>Avg</th>
<th>Worse</th>
<th>Time</th>
<th>Best</th>
<th>Avg</th>
<th>Worse</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pr76</td>
<td>None</td>
<td>157,590</td>
<td>168,840</td>
<td>172,820</td>
<td>4.20</td>
<td>208,559</td>
<td>220,259</td>
<td>258,054</td>
<td>5.02</td>
<td>242,827</td>
<td>272,736</td>
<td>296,349</td>
<td>5.30</td>
</tr>
<tr>
<td></td>
<td>BaB</td>
<td>162,722</td>
<td>167,697</td>
<td>172,089</td>
<td>6.60</td>
<td>205,042</td>
<td>225,023</td>
<td>243,357</td>
<td>7.66</td>
<td>252,958</td>
<td>272,145</td>
<td>289,329</td>
<td>7.80</td>
</tr>
<tr>
<td></td>
<td>CE</td>
<td>153,880</td>
<td>164,661</td>
<td>171,279</td>
<td>4.90</td>
<td>179,156</td>
<td>186,317</td>
<td>190,640</td>
<td>6.65</td>
<td>219,246</td>
<td>226,224</td>
<td>233,538</td>
<td>6.54</td>
</tr>
<tr>
<td></td>
<td>CE+BaB</td>
<td>152,278</td>
<td>166,138</td>
<td>170,752</td>
<td>6.80</td>
<td>177,806</td>
<td>182,381</td>
<td>188,570</td>
<td>8.29</td>
<td>218,901</td>
<td>223,927</td>
<td>229,559</td>
<td>8.77</td>
</tr>
<tr>
<td>Pr152</td>
<td>None</td>
<td>123,907</td>
<td>138,088</td>
<td>143,775</td>
<td>16.70</td>
<td>205,244</td>
<td>228,736</td>
<td>244,971</td>
<td>19.10</td>
<td>249,601</td>
<td>289,744</td>
<td>311,514</td>
<td>19.13</td>
</tr>
<tr>
<td></td>
<td>BaB</td>
<td>124,714</td>
<td>131,109</td>
<td>137,615</td>
<td>27.00</td>
<td>221,838</td>
<td>234,944</td>
<td>244,843</td>
<td>27.80</td>
<td>275,591</td>
<td>304,915</td>
<td>335,748</td>
<td>28.11</td>
</tr>
<tr>
<td></td>
<td>CE+BaB</td>
<td>116,620</td>
<td>131,674</td>
<td>138,091</td>
<td>28.00</td>
<td>132,917</td>
<td>141,993</td>
<td>156,247</td>
<td>32.14</td>
<td>156,131</td>
<td>164,321</td>
<td>174,800</td>
<td>29.49</td>
</tr>
<tr>
<td>Pr226</td>
<td>None</td>
<td>152,016</td>
<td>156,893</td>
<td>163,805</td>
<td>46.70</td>
<td>224,132</td>
<td>247,699</td>
<td>270,164</td>
<td>51.95</td>
<td>284,113</td>
<td>324,268</td>
<td>343,743</td>
<td>51.12</td>
</tr>
<tr>
<td></td>
<td>BaB</td>
<td>149,068</td>
<td>155,574</td>
<td>159,439</td>
<td>77.00</td>
<td>220,032</td>
<td>251,155</td>
<td>272,261</td>
<td>81.43</td>
<td>308,117</td>
<td>340,139</td>
<td>363,043</td>
<td>81.81</td>
</tr>
<tr>
<td></td>
<td>CE</td>
<td>153,409</td>
<td>157,120</td>
<td>163,848</td>
<td>52.60</td>
<td>168,428</td>
<td>172,193</td>
<td>177,955</td>
<td>58.84</td>
<td>181,235</td>
<td>188,813</td>
<td>194,446</td>
<td>58.74</td>
</tr>
<tr>
<td></td>
<td>CE+BaB</td>
<td>148,040</td>
<td>156,629</td>
<td>162,329</td>
<td>75.60</td>
<td>167,782</td>
<td>171,338</td>
<td>175,371</td>
<td>85.43</td>
<td>180,431</td>
<td>188,489</td>
<td>194,800</td>
<td>82.13</td>
</tr>
<tr>
<td>Pr299</td>
<td>None</td>
<td>76,469</td>
<td>78,872</td>
<td>80,880</td>
<td>78.00</td>
<td>116,104</td>
<td>121,429</td>
<td>123,826</td>
<td>82.01</td>
<td>156,812</td>
<td>163,144</td>
<td>168,464</td>
<td>83.17</td>
</tr>
<tr>
<td></td>
<td>BaB</td>
<td>73,177</td>
<td>77,676</td>
<td>80,093</td>
<td>120.90</td>
<td>118,797</td>
<td>121,983</td>
<td>123,266</td>
<td>117.67</td>
<td>152,991</td>
<td>160,755</td>
<td>170,830</td>
<td>123.88</td>
</tr>
<tr>
<td></td>
<td>CE</td>
<td>75,957</td>
<td>78,217</td>
<td>80,945</td>
<td>86.90</td>
<td>76,924</td>
<td>81,323</td>
<td>87,477</td>
<td>95.71</td>
<td>84,266</td>
<td>87,490</td>
<td>94,649</td>
<td>101.94</td>
</tr>
<tr>
<td></td>
<td>CE+BaB</td>
<td>75,145</td>
<td>77,413</td>
<td>78,793</td>
<td>123.10</td>
<td>75,450</td>
<td>78,999</td>
<td>83,613</td>
<td>132.31</td>
<td>85,515</td>
<td>88,526</td>
<td>91,800</td>
<td>134.14</td>
</tr>
<tr>
<td>Pr439</td>
<td>None</td>
<td>146,793</td>
<td>152,224</td>
<td>158,832</td>
<td>184.70</td>
<td>204,525</td>
<td>209,376</td>
<td>217,517</td>
<td>185.17</td>
<td>264,348</td>
<td>270,185</td>
<td>280,318</td>
<td>209.93</td>
</tr>
<tr>
<td></td>
<td>BaB</td>
<td>141,951</td>
<td>146,436</td>
<td>149,874</td>
<td>300.00</td>
<td>202,714</td>
<td>207,095</td>
<td>212,185</td>
<td>283.39</td>
<td>258,487</td>
<td>267,122</td>
<td>274,980</td>
<td>279.41</td>
</tr>
<tr>
<td></td>
<td>CE</td>
<td>142,800</td>
<td>148,416</td>
<td>154,342</td>
<td>215.20</td>
<td>146,162</td>
<td>151,636</td>
<td>157,471</td>
<td>211.26</td>
<td>155,004</td>
<td>159,609</td>
<td>167,336</td>
<td>211.58</td>
</tr>
<tr>
<td></td>
<td>CE+BaB</td>
<td>141,180</td>
<td>147,389</td>
<td>151,975</td>
<td>295.70</td>
<td>144,527</td>
<td>151,392</td>
<td>160,634</td>
<td>301.25</td>
<td>149,649</td>
<td>155,512</td>
<td>161,226</td>
<td>315.70</td>
</tr>
<tr>
<td>Pr1002</td>
<td>None</td>
<td>358,316</td>
<td>375,882</td>
<td>400,960</td>
<td>793.70</td>
<td>441,641</td>
<td>449,855</td>
<td>461,639</td>
<td>800.90</td>
<td>538,708</td>
<td>550,605</td>
<td>558,926</td>
<td>812.93</td>
</tr>
<tr>
<td></td>
<td>BaB</td>
<td>339,890</td>
<td>348,712</td>
<td>365,834</td>
<td>1,154.80</td>
<td>425,578</td>
<td>435,109</td>
<td>444,763</td>
<td>1,182.24</td>
<td>529,951</td>
<td>535,035</td>
<td>540,143</td>
<td>1,149.55</td>
</tr>
<tr>
<td></td>
<td>CE</td>
<td>341,752</td>
<td>349,258</td>
<td>358,935</td>
<td>864.30</td>
<td>366,273</td>
<td>371,649</td>
<td>378,521</td>
<td>894.75</td>
<td>386,697</td>
<td>393,286</td>
<td>400,016</td>
<td>888.89</td>
</tr>
<tr>
<td></td>
<td>CE+BaB</td>
<td>332,652</td>
<td>338,580</td>
<td>346,574</td>
<td>1,224.50</td>
<td>347,126</td>
<td>360,284</td>
<td>368,582</td>
<td>1,298.28</td>
<td>379,677</td>
<td>383,360</td>
<td>389,104</td>
<td>1,240.62</td>
</tr>
</tbody>
</table>
4.3. Comparing with Existing Algorithms

In this section, we compare the performance of the proposed algorithm - GAL against two other existing approaches - Modified Ant Colony Optimization (MACO)\(^1\) and Elite Sweep line Ant Colony Optimization (EACO)\(^2\). Both approaches use Ant Colony Optimization (ACO) to handle MTSP. MACO solves the MTSP by ACO with the ability constraint, where EACO applies Sweep Line Algorithm to improve the initial paths, and uses 3opt local operator to speed up the algorithm. Both of them claimed that their results are the state-of-the-art in the benchmark problems. In this experiment, the parameters setting are listed in Table 2. The algorithm stops when there is no improvement on the fitness value of the best individual in the successive 10000 generations, which means our algorithm reaches the convergent situation.

As those papers only include the case with 5 salesmen, the experiment in this section will also be limited to 5 salesmen only. The experimental results are shown in Table 4. Comparing to existing approaches, GAL can obtain the best fitness. The average fitness values are also greatly improved in the Pr226, Pr299, Pr439 and Pr1002 cases, which are the 4 largest problems. Our approach is always faster than MACO. Although the running time of our method is slightly longer than EACO, the average fitness has obvious improvement in comparison (9.62%).

5. Discussion

In the experiments, we found that inter-crosses could exist in a better solution. An example is shown in Figures 15 and 16. Although no inter-cross is found in the paths of Figure 16, the fitness is 10.07% worse than the path of Figure 15. We suspect this is due to the constraint of maximum cities visited per salesman. From the observation, elimination of inter-cross may not always be a good idea.

Hence, the timing for applying this local operator should be carefully selected. For example, when the solution chromosome is stuck for some generations, this operator can instantly improve the chromosome quality. On the other hand, maintaining the diversity of population reduces the chance of getting into this sticking situation. In addition, with the BaB operator, locally optimal solution has a chance to jump out of the local optimum. The CE operator is safe to use with such precautions in place.

Although the GA\(^3\) described in that paper also handles the same problem instance, it requires pre-computed initial population from a TSP path using Lin and Kernighan operators\(^4\). Our proposed algorithm is suitable for making solution from scratch, in which pre-computed information is not available. Hence, the algorithm is not comparable with our proposed work.
Table 4. Comparison of the proposed algorithm and the existing MTSP solving algorithm with 5 salesmen

<table>
<thead>
<tr>
<th>Problem size</th>
<th>Algorithm</th>
<th>Best fitness</th>
<th>Average fitness</th>
<th>Time (in seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>76</td>
<td>GAL</td>
<td>153389.9</td>
<td>162810.6</td>
<td>17.7</td>
</tr>
<tr>
<td></td>
<td>EACO</td>
<td>157495</td>
<td>157562</td>
<td>19</td>
</tr>
<tr>
<td></td>
<td>MACO</td>
<td>178597</td>
<td>180690</td>
<td>51</td>
</tr>
<tr>
<td>152</td>
<td>GAL</td>
<td>115873.8</td>
<td>128053.4</td>
<td>47.4</td>
</tr>
<tr>
<td></td>
<td>EACO</td>
<td>127791</td>
<td>128004</td>
<td>41</td>
</tr>
<tr>
<td></td>
<td>MACO</td>
<td>130953</td>
<td>136341</td>
<td>128</td>
</tr>
<tr>
<td>226</td>
<td>GAL</td>
<td>148050.6</td>
<td>156542.3</td>
<td>76.6</td>
</tr>
<tr>
<td></td>
<td>EACO</td>
<td>167665</td>
<td>168156</td>
<td>62</td>
</tr>
<tr>
<td></td>
<td>MACO</td>
<td>167646</td>
<td>170877</td>
<td>143</td>
</tr>
<tr>
<td>299</td>
<td>GAL</td>
<td>72949.3</td>
<td>77481.6</td>
<td>108</td>
</tr>
<tr>
<td></td>
<td>EACO</td>
<td>81998</td>
<td>82195</td>
<td>65</td>
</tr>
<tr>
<td></td>
<td>MACO</td>
<td>82106</td>
<td>83845</td>
<td>288</td>
</tr>
<tr>
<td>439</td>
<td>GAL</td>
<td>143785.4</td>
<td>147710.7</td>
<td>269.5</td>
</tr>
<tr>
<td></td>
<td>EACO</td>
<td>161725</td>
<td>162657</td>
<td>95</td>
</tr>
<tr>
<td></td>
<td>MACO</td>
<td>161955</td>
<td>165035</td>
<td>563</td>
</tr>
<tr>
<td>1002</td>
<td>GAL</td>
<td>334350.6</td>
<td>341303.9</td>
<td>1524</td>
</tr>
<tr>
<td></td>
<td>EACO</td>
<td>379871</td>
<td>381654</td>
<td>186</td>
</tr>
<tr>
<td></td>
<td>MACO</td>
<td>382198</td>
<td>387205</td>
<td>2620</td>
</tr>
</tbody>
</table>

6. Conclusion

In this paper, a novel MTSP solving algorithm, called Genetic Algorithm with Local operators (GAL) has been proposed and developed. The new local operators BaB and CE have been successfully deployed to generate high quality results in a short computation time. We have also compared the performance of our work with those of the existing approaches. Our algorithm has made improvement in the search ability and speed.

For future study, as GA can be run in parallel, the proposed method can be further improved by using parallel GA design for real time applications. Furthermore, the local operators can be applied to other variations of MTSP-like problems, like min-max MTSP and multiple objective MTSP to find better solution efficiently.

Acknowledgment(s)

This research was supported by the Vice-Chancellor’s one-off support of The Chinese University of Hong Kong. The authors would like to thank Leung-Yau Lo and Kwan-Yau Cheung for their assistance.


