SOLVING TRAVELING SALESMAN PROBLEMS USING HEURISTIC LEARNING APPROACH

Sim Kim Lau[1], Li-Yen Shue[2]

[1]Department of Information Systems, University of Wollongong, Australia (simlau@uow.edu.au)
[2]Department of Information Management, National Kaohsiung First University of Science and Technology, Taiwan R.O.C (ly_shue@ccms.nkfust.edu.tw)

Abstract

This paper describes the application of a real time admissible heuristic learning algorithm that allows the tour configuration to change as the tour is built. We present a state space transformation process to transform the traveling salesman problem into a state space model. A state space transformation process that defines state, state transition operator and state transition cost will be given. The heuristic evaluation function of the algorithm considers both local and global estimated distance information. The heuristic estimation of a state is computed using minimal spanning tree. The heuristic learning mechanism of this approach allows the heuristic estimates of visited states to be updated, and hence modify the tour configuration along the search process.

1. Introduction

In the traveling salesman problem (TSP), a salesman is to find the shortest tour of a finite number of cities by visiting each city exactly once and returning to the starting city. TSP is inherently intractable. It belongs to a group of problems known as NP-complete. The combinatorial nature of the problem results in computational time to grow exponentially with problem size, and no efficient algorithm could be constructed to find optimal solution in polynomial time [3]. The nature of NP-completeness in TSP makes it unlikely that any algorithm can be guaranteed to find optimal solutions when the number of cities is large [7]. In view of the computation difficulties, various heuristics have been developed to solve the problem. Extensive research has focussed on designing and developing methods that are capable of improving tour so that an optimal solution is obtained. However, there has been relatively little research on using tour construction heuristics to obtain optimal solution. In fact, tour that is built using tour construction heuristics often needs to be improved using tour improvement heuristics so that a better solution can be found. The poor quality of solution obtained using tour construction heuristics can be attributed to these algorithms being greedy approaches. The tour configuration does not change during the tour construction process and the tour is often constructed using local distance information. On the other hand, the key element of local search heuristics is to exchange the edges of the tour in order to change the tour configuration, which allows a better solution to be found.

This paper aims to demonstrate that heuristic learning algorithm in artificial intelligence may present another approach in addressing TSP. A heuristic learning algorithm is applied so that a dynamic tour construction process that allows the tour configuration to change during the tour construction process can be developed. As the tour is constructed, the tour configuration changes through the heuristic learning process using the local and estimated global distance information of the tour. The heuristic learning process allows the heuristic estimates of the partially completed tour to be updated, and, at the same time, the forward search and backtracking operations in the algorithm allow addition and deletion of cities to and from the tour during the tour construction process.

The combinatorial nature of the problem can result in the solution space to become exponential in relation to the problem size. Therefore, it is important to control the search space. This paper also discusses the development of an efficient search strategy that exploits the geometric properties of the cities using computational geometry concept of Delaunay triangulation.

This paper is organised as follows. Section 2 discusses dynamic tour construction using heuristic learning algorithm called Search and Learning A* algorithm. Implementation issues that include state space transformation, heuristic estimation and search strategy are also presented. An example is included in this section to explain the procedure of this approach. Section 3 presents the computational results of randomly generated problems and discusses major factor that influences the performance of this approach. Section 4 presents the conclusions and Section 5 proposes future research direction.

2. Dynamic tour construction using heuristic learning approach

A heuristic learning algorithm called Search and Learning A* algorithm (SLA*) is used for dynamic tour construction. The main feature of SLA* is the application of heuristic learning to update and improve the initially under-estimated heuristic estimates. This will lead to the improvement of state selection decisions and an optimal solution when the goal state is reached. Before SLA* can be applied, the problem needs to be transformed to a state-

space representation. This section introduces the concept of SLA* and discusses the state-space transformation process, the computation of non-overestimating heuristic estimate and a search strategy using Delaunay triangulations.

2.1 Search and Learning A* algorithm

SLA* is a real time admissible heuristic learning algorithm, which uses the heuristic evaluation function to estimate the relative merit of different states to the goal state [11]. The search path improvement feature of SLA* requires the initial estimate of a state to be non-overestimating so that it may be improved during the search process. The rationality of this algorithm is that a state that is further away from the goal state should have a larger heuristic estimate.

Let a front state be x. The state selection process of SLA* is based on the minimum increment of the heuristic function $f(y) = k(x,y) + h(y)$, where $k(x,y)$ is the positive true edge cost from state x to its neighbouring state y, and $h(y)$ is the heuristic estimate of state y. The selection is based on the state with the minimum value of $f(y)$, and then compares it with $h(x)$ to decide if $h(x)$ can be improved. If it does, heuristic learning is said to occur and backtracking will take place. This means the heuristic estimate of state x is too much underestimated and can be updated with the minimum value of $f(y)$. Otherwise the state with minimum $f(y)$ is added to the search path as the new front state and the algorithm continues with the forward search operation.

Due to the fact that the selection criterion of a state is based on the heuristic estimate of its neighbouring states, the update of $h(x)$ with $f(y)$, which is a larger value than before, may invalidate the selection of its previous state (x-1). In order to reflect the effect of the new $h(x)$ to the search path, the algorithm conducts a backtracking operation by applying the above rationale to the state (x-1) to see if $h(x-1)$ can be updated. If $h(x-1)$ is updated, then its previous state (x-2) will need to be reviewed too. In this way, the states of the search path will be reviewed one by one in the reverse order. Along the way, any state whose heuristic estimate has been improved is detached from the path, because the improvement casts doubt on the validity of its previous minimum heuristic function status. This review process continues until it reaches either the state whose heuristic estimate remains unchanged after it has been examined for heuristic learning or the root state, and then the algorithm resumes the search from this state. As a result, before the resumption of the search path, the algorithm would have completely updated the earlier path. Hence the search path that is to be developed subsequently, before the next heuristic learning occurs, will be a minimum path. When the goal state is reached, the path is an optimal path and it represents a complete solution.

2.2 Implementation issues

This section discusses the implementation issues of SLA* on Euclidean TSP, where cities are given as points in a two-dimensional plane and their distance is computed using Euclidean distance. In order to apply SLA*, a state-space transformation method that can formulate the problem into state-space problem must first be developed. A state-space transformation approach, which consists of state definition, state transition operator and state transition cost, will be examined. Next, a suitable method to compute the heuristic estimation will be discussed. Finally, a search strategy that utilises geometric properties of TSP is discussed.

(1) State space transformation approach

The state space transformation process includes state definition, state transition operator and state transition cost.

Definition of state

A state is defined as a tour, consisting of selected cities and the remaining unvisited cities of a given problem. The selected cities form a partially incomplete tour, which is a closed tour by connecting cities in the order of their selection and connecting the last city to the city of origin. A partial tour becomes a complete tour when all cities are included.

State transition

For a given state S_i, $\{(1,2,\ldots,i,1),(U)\}$, where i is the last city added to the partial tour, and U is the remaining unvisited cities, the transition to the next state S_{i+1} is through the selection of a city from the neighbouring cities of i, which are in the unvisited set U. The selection criterion is the minimum increment of tour length.

State transition cost

The transition cost from a parent state S_i to a child state S_{i+1} is the Euclidean distance between two cities.

(2) Heuristic estimation of each state

Among the possible lower bound estimates of a partial tour, minimal spanning tree is used to compute the non-overestimating heuristic estimate. A minimal spanning tree is a spanning tree that connects all nodes such that its cost is minimum. Thus for a state S_i, the heuristic estimate is the cost of the minimal spanning tree on the remaining unvisited cities.

(3) Search strategy

The size of the solution space of TSP is exponential in term of problem size, therefore it is necessary to control the search space so that it is not rapidly becoming too large and an efficient search strategy is very important. Otherwise, the number of possible tours to be considered will increase rapidly. The rationale is to consider only those edges that are likely to result in an optimal tour. The geometric properties of the point sets will be exploited for this purpose. One of the computational geometry concepts that can be used to obtain information about the structure of point sets is Delaunay triangulation.

Delaunay triangulation is the geometric dual of Voronoi diagram. The concept of Voronoi diagram is based on proximity of points in the plane. It partitions the plane into a set of polygons, called Voronoi regions, so that each polygon consists of points closer to one than to any others. Voronoi diagram can be used to solve the nearest neighbour problem, and it allows the proximity question to be answered. The duality of Voronoi diagram and Delaunay triangulation implies that the edges of Delaunay triangulation are orthogonal to their corresponding Voronoi edges [1]. Although in general Delaunay triangulation does not contain a traveling salesman tour, it has been shown that there is a high probability that the edges appear in the optimal tour of TSP are also edges of Delaunay triangulation ([1], [5], [8], [9]). Therefore one can utilise Delaunay triangulation as a search strategy to locate promising neighbouring city that will lead to optimal tour.

2.3 The algorithm for SLA*-TSP

This section describes SLA* algorithm that takes account of Delaunay triangulation as part of the search strategy. This algorithm was developed using the state space transformation process and the heuristic estimation approach using minimal spanning tree identified in the previous section. In the algorithm, the completed tour length for state S_i is computed rather than the heuristic estimate of S_i. This allows the estimated tour length to be compared. However, the result is the same as that when the heuristic estimate is computed. The algorithm, acronym as SLA*-TSP (Search and Learning Algorithm for Traveling Salesman problems), is given as follows:

Let S_i be the i^{th} state with its tour $P_i(1,2,...,i,1)$, where 1 is the city of origin and i is the last city of the tour. Its heuristic estimate h(i) is the minimum spanning tree of the remaining (n-i) cities. S_i is the goal state when i = n.
d(i,j) be the Euclidean distance between city i and city j.
H(i) be the estimated tour length for S_i, which consists of the tour p_i and h(i).

Step 0: Apply Delaunay triangulation algorithm to find neighbouring nodes for each city.

Step 1: Locate the city of origin as the one with the smallest x-coordinate; choose the city with the largest y-coordinate to break ties.

Step 2: Put the root state on the backtrack list called OPEN.

Step 3: Call the top-most state on the OPEN list S_i. If S_i is the goal state, stop. Otherwise continue.

Step 4: Find the $(i+1)^{th}$ city with $\min\{[d(1,2)+d(2,3)+\ldots+d(i-1,i)+d(i,i+1)+d(i+1,1)] +h(i+1)\}$ from neighbouring cities of i; break ties randomly. If no neighbouring city of i can be found, go to step 6.

Step 5: If $\{[d(1,2)+d(2,3)+\ldots+d(i-1,i)+d(i,1)] + h(i)\} \geq \min\{[d(1,2)+d(2,3)+\ldots+d(i-1,i)+d(i,i+1)+d(i+1,1)] + h(i+1)\}$, add S_{i+1} to the OPEN list as the top-most state; otherwise replace h(i) with $[d(i,i+1)+d(i+1,1) + h(i+1) - d(i,1)]$.

Step 6: Remove S_i from OPEN list if it is not the root state.

Step 7: Go to step 3.

2.4 Example

An example of an 8-city problem obtained from [4] (see Figure 1) is included in this section to demonstrate the working of SLA*-TSP. This section describes the procedures when the algorithm is applied. The complete search process in finding an optimal solution is summarized in Table 1. In the table, each row represents one attempt to locate a tour as far as possible without incurring heuristic learning. The search path in each row is disrupted as soon as the estimated tour length of a newly selected state is greater than that of its parent state. Thus the last state of each row is the one that has a larger estimated tour length and causes the backtracking. When this happens, the algorithm will update the estimated tour length of its parent state. The upper entry in each cell represents a state, and the lower entry is the estimated tour length of that state. The search process consists of both forward searching and backward updating. In order to maintain the simplicity of the table, only the forward searching part in each row is presented. The heuristic updating and the backtracking process of the algorithm is not shown, instead an asterisk ("*") is used to indicate the state where backtracking ends and the new round of forward search begins. The following describes the steps and procedures when SLA*-TSP is applied.

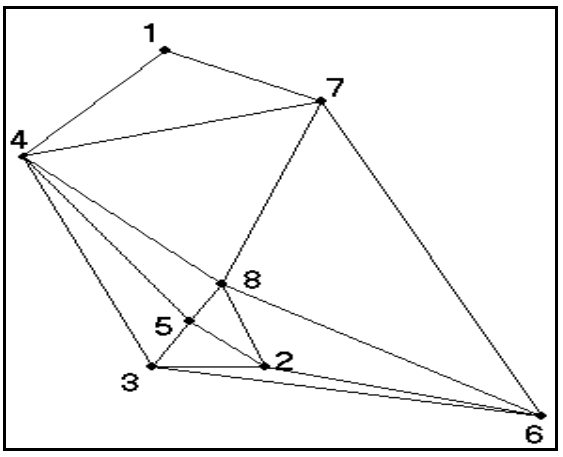

Figure 1 An example of 8-city problem

Initially the neighbouring edges for each city in the problem are identified using Delaunay triangulation. In this example, the city of origin is selected using the minimum x-coordinate. Therefore the city of origin is city 4: the corresponding root state is {(4,4),U}, and its heuristic estimate to the goal state is the minimal spanning tree of the remaining unvisited cities U (i.e. cities 1,2,3,5,6,7,8), which is 13070. For simplicity, from hereafter only the tour part of a state is shown, and its unvisited states U is not given. The selection of the next city to join the tour is made from the five neighbouring cities (1,3,5,7,8) of city 4 obtained from the edges of Delaunay triangulation. Thus the corresponding front states are: (4,1,4), (4,3,4), (4,5,4), (4,7,4) and (4,8,4). Step 4 of the algorithm makes the selection by finding the state with the smallest estimated tour length. As shown in row 1, state (4,1,4) is selected with a smaller value of 15891 = 10785 (heuristic) + 5106 (tour), which is greater than 13070 of its parent state (4,4). Step 5 of the algorithm updates the parent state with this new value 15891, and backtracks to its parent state (4,4).

In row 2, the algorithm starts from (4,4). It again selects (4,1,4), and because of no heuristic updating involved, the state (4,1,4) becomes the new front state. For the state selection of the next round, the candidate edge set shows that promising neighbouring cities include 1, 4 and 7, as shown in Figure 1. Since the last city that was added to the tour is 1, then the states to be considered are those of its neighbouring cities 4 and 7. City 4 is not eligible because it was in the tour already. Hence, the next state to consider is (4,1,7,4). Its estimated tour length is 15740 = 7154 (heuristic) + 8586 (tour). This value is not greater than 15891 of its parent state, hence state (4,1,7,4) becomes the new front state. From city 7 (which was the last city added to the partial tour), only cities 6 and 8 are eligible for consideration. The estimated tour lengths of these two corresponding states are 23093 and 18548 respectively. State (4,1,7,8,4) is selected for having a smaller value of 18548. However, this value is larger than 15740 of its parent state (4,1,7,4). Hence, the estimated tour length of (4,1,7,4) is updated to 18548, and the algorithm backtracks to (4,1,4). From (4,1,4), the state (4,1,7,4) is selected with an estimated tour length of 18548, which updates that of (4,1,4) to 18548. The algorithm then backtracks to the root state (4,4). The selection of state (4,1,4) leads to the update of the estimated tour length of state (4,4) from 15891 to 18548.

Row 3 shows the search from the newly updated state (4,4). The search path shows states (4,1,4), (4,1,7,4), (4,1,7,8,4), (4,1,7,8,5,4), (4,1,7,8,5,3,4), (4,1,7,8,5,3,2,4) and stops at (4,1,7,8,5,3,2,6,4). At state (4,1,7,8,5,3,2,6,4), its estimated tour length 24122 is greater than 16964 of its parent state (4,1,7,8,5,3,2,4). This causes the algorithm to backtrack. The algorithm backtracks all the way to the root state (4,4), and its estimated tour length is updated to 19348. With this new value, row 4 shows the next search path.

In row 4, there are three possible child states: (4,7,1,4), (4,7,6,4) and (4,7,8,4). The state (4,7,1,4) is pruned from consideration because city 1 does not have an unvisited neighbouring city to allow the state generation to continue. Thus, there are only two possible child states: (4,7,6,4) and (4,7,8,4). It is found that state (4,7,8,4) with tour estimated tour length of 22569 is the state with the minimum estimated tour length. This value is greater than 19348 of its parent state, and causes the algorithm to backtrack. The algorithm backtracks to the root state (4,4), and its estimated tour length is updated to 19348. With this new value, row 5 shows the next search path.

The same procedure is followed and steps are repeated until there is no more backtracking and heuristic learning, in which case optimal solution is found. Table 1 shows the summarised search process. The table shows that the algorithm takes 19 forward search trials to reach an optimal solution of 22300 with the optimal tour being (4,8,5,3,2,6,7,1,4).

Table 1 Summarised search process for 8-city problem

Iteration	Root state	Level-1	Level-2	Level-3	Level-4	Level-5	Level-6	Level-7
1	(4,4) * 13070	(4,1,4) 15891						
2	(4,4)* 15891	(4,1,4) 15891	(4,1,7,4) 15740	(4,1,7,8,4) 18548				
3	(4,4)* 18548	(4,1,4) 18548	(4,1,7,4) 18548	(4,1,7,8,4) 18548	(4,1,7,8,5,4) 18464	(4,1,7,8,5,3,4) 18409	(4,1,7,8,5,3,2,4) 16964	(4,1,7,8,5,3,2,6,4) 24122
4	(4,4)* 19348	(4,7,4) 19348	(4,7,8,4) 22569					
5	(4,4)* 20415	(4,5,4) 20415	(4,5,3,4) 20360	(4,5,3,2,4) 22171				
6	(4,4)* 20424	(4,8,4) 20424	(4,8,5,4) 21037					
7	(4,4)* 20740	(4,3,4) 20740	(4,3,5,4) 20360	(4,3,5,8,4) 20982				
8	(4,4)* 20982	(4,3,4) 20982	(4,3,5,4) 20982	(4,3,5,8,4) 20982	(4,3,5,8,2,4) 21650			
9	(4,4)* 21037	(4,5,4) 21037	(4,5,8,4) 21037	(4,5,8,7,4) 23310				
10	(4,4) 21037	(4,8,4)* 21037	(4,8,5,4) 21037	(4,8,5,3,4) 20982	(4,8,5,3,2,4) 21054			
11	(4,4)* 21054	(4,8,4) 21054	(4,8,5,4) 21054	(4,8,5,3,4) 21054	(4,8,5,3,2,4) 21054	(4,8,5,3,2,6,4) 21678		
12	(4,4)* 21429	(4,1,4) 21429	(4,1,7,4) 21429	(4,1,7,8,4) 21429	(4,1,7,8,2,4) 21429	(4,1,7,8,2,5,4) 20744	(4,1,7,8,2,5,3,4) 16793	(4,1,7,8,2,5,3,6,4) 26402
13	(4,4) 21429	(4,1,4) 21429	(4,1,7,4) 21429	(4,1,7,8,4) 21429	(4,1,7,8,2,4)* 21429	(4,1,7,8,2,3,4)* 21142	(4,1,7,8,2,3,5,4) 16263	(4,1,7,8,2,3,5,6,4) 26236
14	(4,4)* 21445	(4,1,4) 21445	(4,1,7,4) 21445	(4,1,7,8,4) 21445	(4,1,7,8,5,4) 21445	(4,1,7,8,5,2,4) 21445	(4,1,7,8,5,2,3,4) 16643	(4,1,7,8,5,2,3,6,4) 26252
15	(4,4)* 21650	(4,3,4) 21650	(4,3,5,4) 21650	(4,3,5,8,4) 21650	(4,3,5,8,2,4) 21650	(4,3,5,8,2,6,4) 22274		
16	(4,4)* 21678	(4,8,4) 21678	(4,8,5,4) 21678	(4,8,5,3,4) 21678	(4,8,5,3,2,4) 21678	(4,8,5,3,2,6,4) 21678	(4,8,5,3,2,6,7,4) 21210	(4,8,5,3,2,6,7,1,4) 22300
17	(4,4)* 22171	(4,5,4) 22171	(4,5,3,4) 22171	(4,5,3,2,4) 22171	(4,5,3,2,8,4) 20353	(4,5,3,2,8,6,4) 23384		
18	(4,4)* 22274	(4,3,4) 22274	(4,3,5,4) 22274	(4,3,5,8,4) 22274	(4,3,5,8,2,4) 22274	(4,3,5,8,2,6,4) 22274	(4,3,5,8,2,6,7,4) 21806	(4,3,5,8,2,6,7,1,4) 28896
19	(4,4)* 22300	(4,8,4) 22300	(4,8,5,4) 22300	(4,8,5,3,4) 22300	(4,8,5,3,2,4) 22300	(4,8,5,3,2,6,4) 22300	(4,8,5,3,2,6,7,4) 22300	(4,8,5,3,2,6,7,1,4) 22300

3. Factor influencing the performance of SLA*-TSP

This section discusses factor that influences the performance of SLA*-TSP. The investigation will be conducted on randomly generated problems. The aim of this experiment is to investigate the performance of SLA*-TSP in relation to the pattern in which the nodes are distributed in the Euclidean plane. Although the experiments were conducted on relatively small size problems, the aim is to analyse the results from these problems and investigate the search behaviour of the approach. Hence, the factor that influences the performance of SLA*-TSP algorithm may still be applicable to problems of larger sizes.

3.1 Method

The test approach is adapted from Laporte et al [6], and the test problems consist of nodes located within a (0,100) square. The square is divided into 16 equal rectangles (see Figure 2). Six nodes are randomly generated within each rectangle according to a uniform distribution. The nodes were selected from four rectangles within the square and each test problem consists of twenty-four nodes. Each problem is named using the number matching each rectangle in the square. For example, p1_4_16_13 refers to nodes obtained from rectangles 1, 4, 16 and 13.

4	8	12	16
3	7	11	15
2	6	10	14
1	5	9	13

Figure 2 Structure of problems tested

Five problems have been selected in this experiment: p1_6_11_16, p3_7_11_15, p1_2_3_4, p1_8_9_16, p1_4_16_13. Each problem shows a different characteristic in which the clusters are located in the square. For example, the nodes in problems p1_4_16_13 and p1_8_9_16 show a characteristic of clusters that are dispersed and well separated. On the other hand, the clusters in problems p1_2_3_4, p3_7_11_15 and p1_6_11_16 are located close to one another. In terms of distance between clusters, problems p1_4_16_13 and p1_8_9_16 both demonstrate the distance between clusters (inter-cluster distance) is significant, comparing to the other three problems.

3.2 Results and discussions

This section presents the test results in terms the number of heuristic updates and the quality of the heuristic estimate, which is expressed as the ratio of the initial heuristic estimate of the root state to the optimal solution. The results shown were averaged over ten different instances for each type of problem. Table 2 shows the results obtained when SLA*-TSP is applied. Column 2 shows the number of heuristic updates required, and column 3 shows the ratio between the value of initial heuristic estimate of the root state to the optimal solution found.

Table 2 Experimental results

Problem	Number of heuristic updates	Ratio = initial heuristic estimate of the root state/optimal solution
p1_6_11_16	321201	0.5896
p3_7_11_15	295200	0.6547
p1_2_3_4	257204	0.6280
p1_8_9_16	26527	0.6908
p1_4_16_13	7159	0.7564

The ways nodes are grouped in different clusters can influence Delaunay triangulation. Minimal spanning tree is a subset of Delaunay triangulation [1], thus the behaviour of clustering can influence the quality of the heuristic estimate, which is computed using minimal spanning tree. Figures 2 and 3 show Delaunay triangulation for problems p1_4_16_13 and p1_2_3_4 respectively. These two problems were selected to contrast the Delaunay triangulation formed. The nodes in problem p1_4_16_13 are grouped into distinct and well-separated clusters. The Delaunay triangulation produced can be described as ‘wide’ and ‘fat’. On the other hand, the clusters in problems p1_2_3_4 are close to one another, and the Delaunay triangulation produced can be described as ‘elongated’, ‘narrow’ and ‘skinny’. The descriptions used to described Delaunay triangulation as ‘fat’ and ‘skinny’ follow the convention used in [1], [2], and [10].

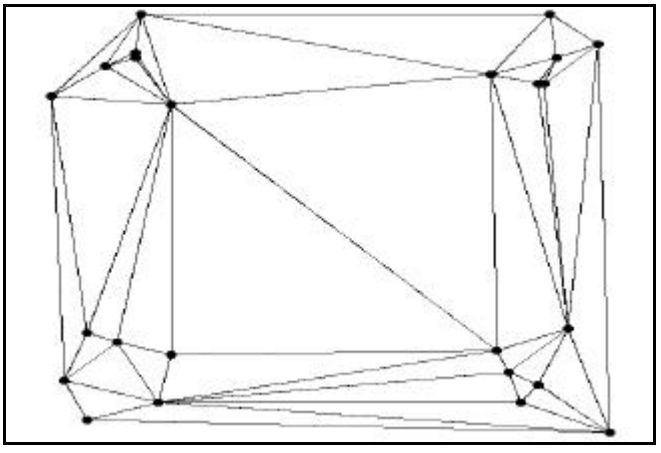

Figure 2 Delaunay triangulation for p1_4_16_13

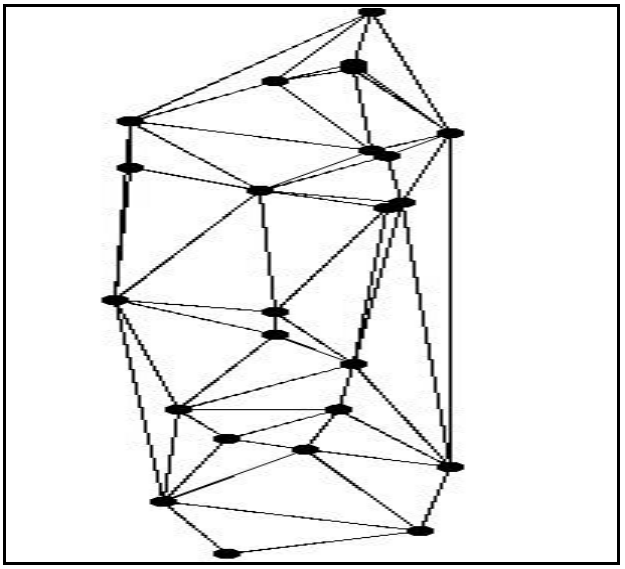

Figure 3 Delaunay triangulation for p1_2_3_4

Column 3 shows that the ratio for problems p1_4_16_13 and p1_8_9_16 is higher, comparing to the other three problems. In addition, the number of heuristic updates for these two problems is comparatively smaller comparing to the other three problems. This result can be explained by the way the nodes are grouped into clusters. The clusters in problems p1_4_16_13 and p1_8_9_16 are more dispersed and well-separated. In contrast, the clusters in each of the problems p1_2_3_4, p1_6_11_16 and p3_7_11_15 are situated near to one another. In problems where clusters are distinct and well-separated, the edges that connect different clusters could be longer. These long edges between the clusters would form part of the minimal spanning tree, because minimal spanning tree must connect all nodes. Therefore, when the clusters are dispersed and located far from one another, it will result in a higher heuristic estimate. On the other hand, in problems where clusters are situated near one another, the edges between clusters are shorter. This could result in a lower value of heuristic estimate. When the heuristic estimate is comparatively low, it will take more heuristic updates and backtracking to reach the optimal solution. Therefore, one can conjecture that SLA*-TSP approach is suitable for problems that exhibit distinct and well-separated clusters. This is because long edges between the clusters will always form part of the edges in minimal spanning tree, and it will result in a higher value of initial heuristic estimate of the root state, which can lead to better performance.

4. Conclusion

In this paper, the main features of SLA*-TSP have been examined. The formulation of a traveling salesman problem into a state space problem is achieved by defining the state, state transition operator and state transition cost. The development of the state-space transformation process enables the heuristic learning algorithm of SLA* to be applied so that the tour can be constructed dynamically. This approach of constructing tour allows the tour configuration to change during the tour construction process. This is made possible by the backtracking and heuristic updating processes, which allow cities to be added and deleted during the tour construction processes. The application of heuristic evaluation function of the algorithm allows both local and global estimated distance information to be used. This way SLA*-TSP is not relying on local knowledge alone to build tour. The example presented in this paper has shown that, the backtracking and forward search processes have repetitively led to the deletion and addition of cities from and to the tour through the consideration of both local and the global estimated distance information.

This paper has also demonstrated that when the geometric properties of Delaunay triangulation is incorporated into the search strategy of SLA*-TSP, the performance of SLA*-TSP is greatly enhanced through the reduction of solution space selection. This is because Delaunay triangulation identifies only promising cities to be considered by a given state, and by ensuring that edges that will not lead to optimal solution are pruned during the search process.

Our results show the main factor that influences the performance of SLA*-TSP is the value of the initial heuristic estimate. If the initial value of the heuristic estimate of the root state is close to the optimal solution, then fewer backtracking and heuristic updates will be required to reach the optimal solution. The way nodes are grouped into clusters in the Euclidean plane also influences the quality of the heuristic estimate. A better performance of SLA*-TSP can be achieved for problems that exhibit distinct and well-separated clusters, because this type of problem produces a higher quality of heuristic estimate. Therefore, the closer the value of the heuristic estimate is to the optimal solution, the better the performance of SLA*-TSP.

5. Future research

The major factor influencing the performance of SLA*-TSP is the heuristic estimate. Minimal spanning tree has been used in this research to compute the heuristic estimate. However, further research can be investigated to find an alternative method, if any, to compute the heuristic estimate that has a value as close to the optimal solution as possible.

References

[1] Aurenhammer, F. 1991. Voronoi Diagrams – A Survey of a Fundamental Geometric Data Structure. *ACM Computing Surveys*, 23(3), 345-405

[2] de Berg, M., van Kreveld, M, Overmars, M. and Schwarzkopf, O. 1997. *Computational Geometry: Algorithms and Applications*, Springer-Verlag, Berlin

[3] Garey N.R. and Johnson, D.S. 1979. *Computers and Intractability: a Guide to the Theory Of NP-Completeness*. W.H Freeman

[4] Gendreau, M., Hertz, A. and Laporte, G. 1992. New Insertion and Postoptimization Procedures for the Traveling Salesman Problem. *Operations Research*, 40(6), 1086-1094

[5] Krasnogor, N., Moscato, P. and Norman, M.G. 1995. A New Hybrid Heuristic for Large Geometric Traveling Salesman Problems Based on the Delaunay Triangulation. *Anales del XXVII Simposio Brasileiro de Pesquisa Operacional*, Vitoria, Brazil, 6-8 Nov 1995

[6] Laporte, G., Potvin, J. and Quilleret, F. 1996. A Tabu Search Heuristic using Genetic Diversification for the Clustered Traveling Salesman Problem. *Journal of Heuristics*, 2, 187-200

[7] Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G. and Shmoys, D.B. 1985. *The Traveling Salesman Problem.* John Wiley & Sons

[8] Phan, C. 2000. Investigating Delaunay Triangulation as a Basis for a Travelling Salesman Problem Algorithm. *Transactions Of The Oklahoma Junior Academy Of Science* (Http://Oas.Okstate.Edu/Ojas/Phan.Htm)

[9] Stewart Jr., W.R. 1992. Euclidean Traveling Salesman Problems and Voronoi Diagrams, presented at *the ORSA-CSTS Conference*, Williamsburg, VA

[10] O'Rourke, J. 1998. *Computational Geometry in C* (second edition), Cambridge University Press

[11] Zamani, M.R. 1995. *Intelligent Graph-Search Techniques: an Application to Project Scheduling Under Multiple Resource Constraints*. Phd Thesis