Using a Divide and Conquer Method for Routing in a PC Vehicle Routing Application

Brenda Cheang
Department of Management Information Systems
University College Dublin
Belfield, Dublin 4, Ireland.

Sherlyn Foo
Systems and Computer Organisation
1 Depot Road, #08-01
Defence Technology Towers
Singapore 109679

Andrew Lim
School of Computing
National University of Singapore
Lower Kent Ridge, Singapore 119260

Abstract

Vehicle routing is an area of both practical and theoretical importance. Unfortunately, routing problems are notably difficult to solve since almost all vehicle routing problems are NP-hard. However, we have discovered that when reduced to its simplest form, solving a single vehicle routing problem, without capacity and time window constraints is quite similar to the Traveling Salesman Problem (TSP). In this paper, we studied various well-known traveling salesman heuristics such as the Nearest Neighbor, the Space-Filling Curve, and the 2-Opt. In addition, we proposed a hybrid that can trade off the quality of a solution with computing time. The input to any TSP heuristic is the all-pairs shortest paths of the points to be visited. We have found the Floyd-Warshall algorithm, a traditional method to compute all-pairs shortest paths, to be too time-consuming, especially for large data sets. Hence, in this paper, we also proposed an algorithm that generates approximate all-pairs shortest paths. Our experimental results showed that our algorithm's errors are quite small. More importantly, these estimated shortest paths information is sufficient for TSP heuristics to produce good tours especially for geometric graphs like road networks. In addition, our approximate all-pairs shortest paths also allowed pre-computation, thus speeding up shortest path computations tremendously. Finally, our algorithms were implemented and integrated into a vehicle routing application that can be run on a PC.

1 Introduction

Vehicle routing [1,2] is an area of both practical and theoretical importance. From the practical point of view, good routing procedure can save the industry millions of dollars per year through cost-effective movement and delivery of goods. In addition, effective routing can increase productivity, improve operations and make the job of the dispatcher easier.

However, routing problems are notoriously difficult to solve since almost all routing problems are NP-hard. One must consider both the nature of a particular routing problem and the environment in which the solution of the problem is being used. Some of such considerations include fleet size, vehicle and crew cost, number of routes run daily, etc.

These considerations, unfortunately, have turned the theoretical analysis of the performance quality of routing problems into a non-trivial task. However, the essence of most variations of routing problems is to minimize the distance-related cost. Hence, in this paper, we focus on achieving this fundamental objective.

We have found that when reduced to its simplest form, solving a single-vehicle routing problem is quite similar to solving the classical Traveling Salesman Problem (TSP). We conducted experiments on several popular TSP heuristics, namely Nearest Neighbor (NN), Space-Filling Curve (SP), and Nearest Neighbor improved by 2-Opt (NN-2Opt). We discovered that NN and SP were able to generate tours rather quickly but at the expense of the quality of results. We also found that although the 2-Opt heuristic generates good tours, it takes too long for it to be useful in real world applications. This led to our proposal of a new heuristic that is able to trade-off between the quality of a solution and computing time.

In order to use any TSP heuristic, the first step is to locate the all-pairs shortest paths among points to be visited. However, current all-pairs shortest paths are too slow. Hence, we propose a clustering technique that can speed up the computation of all-pairs shortest paths; and by performing some pre-computations, we can achieve further speed-ups.

Finally, the programs written specially for this paper are in Visual C++ and the CPU times recorded are in seconds on 486-DX 100 with 16 Mbytes of extended memory.

2 Comparison of TSP Heuristics

We conducted tests on three of the most popular heuristics for TSP, namely the Nearest Neighbor (NN), the Space-Filling Curve (SP), and the Nearest Neighbor improved by 2-Opt (NN-2Opt) to analyze their performances in terms of tour lengths generated as well as their computation times. Table 1 is a summary of the results. For more details of these heuristics, please refer to [3,4].

	Tour Length (% overNN-2-Opt)		Tour Generation Time (s)		
Heuristic Name	NN	SP	NN-2Opt	NN	SP
Problem Size (pt)					
100	123.76	130.58	7	1	1
200	121.55	123.45	50	1	1
500	117.50	121.60	946	3	1
800	118.00	125.30	3400	8	1
1000	117.70	120.20	6536	12	1

Table 1 Tour Lengths and Generation Times Produced by TSP Heuristics

The NN-2Opt heuristic produced the shortest tour length in all test cases. However, we found it too time-consuming, taking O(n^3) time. In contrast, the Space-

Filling Curve technique is very fast, taking just O(*n*log*n*) time. But, the generated tour lengths using SP are relatively poor, even losing out to the NN heuristic for test problem sizes of up to 1000 points. Moreover, the gain in tour generation time using SP over NN is but a matter of a few seconds, even for problems with 1000 points.

3 Partitioning Using NN-2Opt Heuristic

3.1 Simple Partitioning-NN-2Opt Heuristic

We used the "divide and conquer" technique to reduce the computation time of NN-2Opt heuristic. Suppose all the points to be visited are enclosed in a rectangle. This rectangular-pointed coverage is then divided into *n* smaller identical rectangular regions. A partial tour through all the points within each sub-region is constructed using NN-2Opt heuristic. All the partial tours in each region are then linked up. The results revealed that inter-partial tour links improved when the 2-Opt heuristic was implemented (See Figure 1).

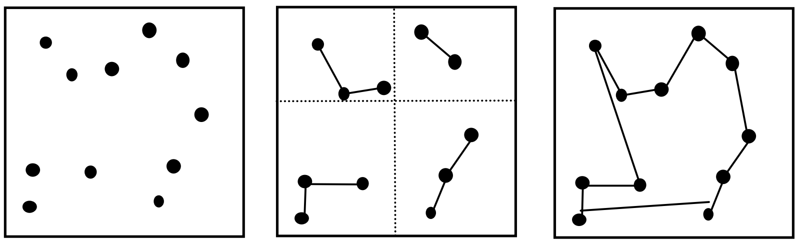

Figure 1 Simple Partitioning-NN-2Opt Heuristic

We observed that the simple Partitioning-NN-2Opt heuristic has a tendency to yield very bad tours when most pairs of end-points of the partial tours are facing away from one another. Figure 2 shows how wrong orientations of partial tours will result in lengthy inter-partial tour links.

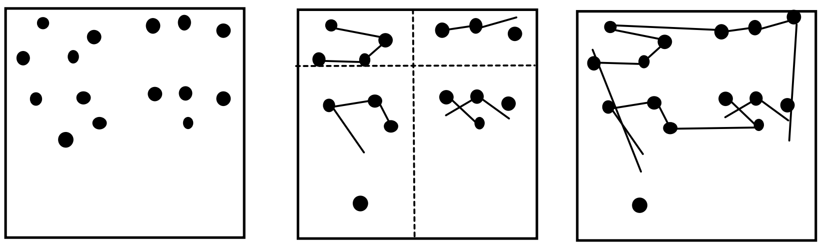

Figure 2 Poor Partitioning-NN-2Opt Heuristic

3.2 Improved Partitioning-NN-2Opt Heuristic

The nature of the Space-Filling Curve technique is that it recursively visits each of the consecutive quadrants of a square in either the clockwise or anti-clockwise direction. This means that the points are visited in either one of these directions. We can use this property of the space-filling curve to guide the orientation of each partial tour so that all pairs of endpoints of each partial tour will face inwards. This way, we can keep inter-partial tour link lengths to the minimum, thus reducing the overall tour length. Figure 3 illustrates how the poor tour shown in Figure 2 can be improved by using the SP heuristic to orientate each partial tour.

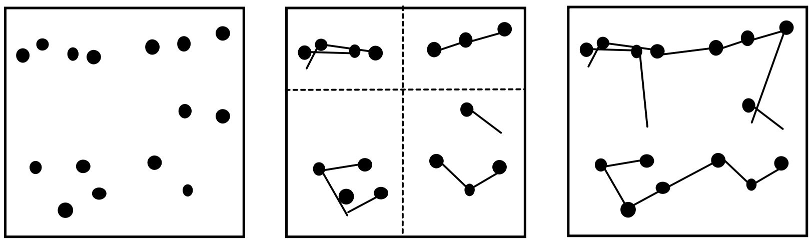

Figure 3 Improved Partitioning-NN-2Opt Heuristic

Basically, our Partitioning-NN-2Opt Heuristic consists of the following 4 main steps:

(i) Dividing the area enclosing the points to be visited into *n* rectangular partitions (where *n* should be a multiple of 2). This is followed by a grouping of these points into their respective partitions based on their positions.

(ii) Applying the Space-Filling Curve technique to each partition of points so as to guide the order of the visitation of points such that the endpoints of each partial tour all face toward their respective centers.
Some of the partitions will lie in odd-numbered columns/rows while others lie in even-numbered columns/rows. The orientation of the space-filling curve used for a particular partition will then depend on whether it lies in an even/odd numbered row and column. An example is shown in Figure 4 where if the partition lies in an even numbered row and column, the order of the visitation of points will be based on their position on a space-filling curve with endpoints facing North-west.

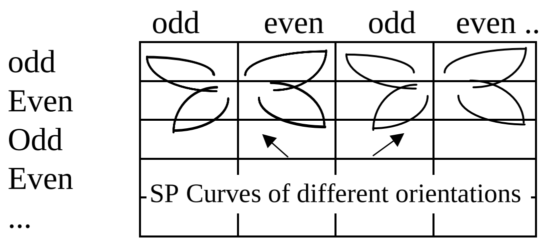

Figure 4 Different Orientations of Space-Filling Curve

(iii) Applying NN-2Opt heuristic to all the points in each partitioned region to construct a partial tour.

(iv) Linking up all the endpoints of each partial tour to form a complete tour and using the 2-Opt heuristic to improve the inter-partial tour links.

3.3 Time Complexity

When a graph of n points are to be visited and these points are well distributed into m partitions, grouping these points into m partitions will take O(n) time. Applying the Space-Filling Curve technique to orientate the partial tours in all partitions takes O($n\log{}^{n}/_{m}$) time. Using the NN-2Opt heuristic to form m partial tours, however, takes O(${}^{n^3}/_{m^2}$) time. Additionally, improving inter-partial tour links using the 2-Opt heuristic will take another O(m^3) amount of time. Thus, the running time of the Partitioning-SP-NN-2Opt heuristic would be

$$O(\max \{ n, n \log {}^{n}/_{m}, {}^{n^3}/_{m^2}, m^3 \}).$$

Hence, we see that a time complexity of O($n^{1.8}$) is achievable if we have $n^{0.6}$ partitions.

3.4 Experimental Results

In this section, we analyze the partitioning-NN-2Opt heuristic, the Partitioning-SP-NN-2Opt heuristic and the NN-2Opt heuristic in terms of their running times and generated tour lengths.

We conducted an experiment in which each heuristic was run with different sizes of test problems. The results are summarized in Tables 2 and 3. Table 2 tabulates the tour lengths while Table 3 tabulates the tour construction times needed by the various heuristics. To facilitate analysis, only the averages of the tour lengths and tour generation times are being shown. All tour lengths are expressed as a percentage over the corresponding tour lengths generated by NN-2Opt.

	Tour Length (% over NN-2Opt)					
	4 Partitions		8 Partitions		(Data Size)$^{0.6}$ Partitions	
Heuristic Name	Partition-NN-2Opt	Partition-SP-NN-2Opt	Partition-NN-2Opt	Partition-SP-NN-2Opt	Partition-NN-2Opt	Partition-SP-NN-2Opt
Data Size						
100	120.58	112.16	121.82	115.76	121.60	120.94
200	117.00	111.73	121.95	115.93	122.78	118.78
500	111.25	104.98	115.25	110.53	126.70	123.50
800	112.97	106.50	112.73	111.50	127.63	123.67
1000	108.90	104.30	110.90	110.00	123.90	119.10

Table 2 Lengths of Tours Constructed with Different Number of Partitions

		Tour Generation Time (s)					
		4 Partitions		8 Partitions		(Data Size)$^{0.6}$ Partitions	
Heuristic Name	NN-2Opt	Partition-NN-2Opt	Partition-SP-NN-2Opt	Partition-NN-2Opt	Partition-SP-NN-2Opt	Partition-NN-2Opt	Partition-SP-NN-2Opt
Data Size							
100	7	1	1	1	1	< 1	< 1
200	50	3	3	1	1	< 1	1
500	946	46	47	13	13	1	2
800	3400	208	212	53	49	3	4
1000	6536	357	364	97	93	3	5

Table 3 Tour Generation Times Needed with Different Number of Partitions

Experimental results show that the Partitioning-SP-NN-2Opt heuristic consistently produces better tours than the Partitioning-NN-2Opt heuristic. The extra time taken by the Partitioning-SP-NN-2Opt heuristic use the SP technique to orientate all partial tours is very small since the SP technique needs O($n\log n$) running time. Thus both heuristics take almost the same time.

We also observed that the 4-Partitioning-SP-NN-2Opt heuristic may take a slightly shorter time than the 4-Partitioning-NN-2Opt heuristic for large test data. A plausible reason may be because the usage of the SP technique in the Partitioning-SP-NN-2Opt heuristic caused the tour to be reasonably good, thus reducing the number of possible 2-Opt swaps needed to reach a 2-Opt optimum.

The results also showed that as the number of partitions used approaches $n^{0.6}$ (where n is the number of points visited), the tour construction time taken gets shorter while the tour quality becomes poorer.

Finally, for a large data set (e.g. 1000 points), the 4-Partitioning-SP-NN-2Opt heuristic has generated rather good tours – approximately 4.3% worse than the NN-2Opt heuristic – and managed to save over thousands of seconds of computation time.

4 Approximate All-Pairs Shortest Paths

In this section, we propose an algorithm which can compute approximate shortest distances between all pairs of points in quadratic time for sparse graphs.

Our algorithm uses the "divide and conquer" approach to speed up running time. Points to be visited are first grouped into clusters. The all-pairs shortest paths among the points within each cluster as well as the all-pairs shortest inter-cluster paths are computed. The shortest path between any two points in two different clusters is assumed to be via their respective shortest inter-cluster path. Thus, approximate all-pairs shortest paths can be computed.

An outline of this approximation algorithm is described below:

I. Construct an edge table to hold direct edge information between points (similar to the first step in Floyd-Warshall's Algorithm).
II. Construct x Minimum Spanning Trees (spanning forest of x trees) to group the points to be visited into x clusters.
III. Use Floyd-Warshall's algorithm to compute all-pairs shortest paths among all the points in each cluster.
IV. Compute inter-cluster direct shortest paths (if one exists).
V. Use Floyd-Warshall's algorithm to compute inter-cluster shortest paths among all pairs of clusters.
VI. Compute approximate all-pairs shortest paths among all the points using information from intra-cluster shortest paths and inter-cluster shortest paths.

4.1 Time Complexity

Let us suppose that there exist n points and e edges in a graph. These n points are divided into $^n/_s$ clusters, each containing an average of s points. Building MSTs take O(e log n) time. Each Intra-Cluster All-Pairs Shortest Path computation takes O(s^3) time. Since there are $^n/_s$ clusters, it will take O(ns^2) time. All Inter-Cluster Direct

Shortest Paths computation take O(e) time, while Inter-Cluster All-Pairs Shortest Paths computation take $O((^n/_s)^3)$ time, and the Approximate All-Pairs Shortest Paths computation take $O((^n/_s) * (^n/_{s} - 1) * s^2)$ time. Thus, the running time of the Approximation All-Pairs Shortest Paths Algorithm is

$O(\max \{ e \log n, ns^2, e, (^n/_s)^3, n^2 \})$.

4.2 Experimental Results

4.2.1 Test Problems

We have carried out our experiments on two types of graphs, namely random and geometric graph.

In a random graph, edges between all pairs of points will be arbitrarily given a probability of existence. If we want a dense random graph, we will set the allowed probability of existence p to be high so that all edges with probability of existence less than p will be considered present. Conversely, for a sparse random graph, we will set p to be low.

In a geometric graph, only edges within certain length will be considered present. We also assume the area enclosing the points to be visited to be a unit square (through suitable scaling).

The bounding length is defined as:

$$\sqrt{\frac{Expected\ No\ of\ Adjacent\ Edges}{(Total\ No\ of\ Po\text{int}\,s * \Pi)}}$$

An edge is present if its length between the 2 points is less than the predefined bounding length. If we want a dense geometric graph, we will set its expected number of adjacent edges to be high. Conversely, for a sparse geometric graph, we will set it to be low.

4.2.2 Estimated Shortest Paths Quality Analysis

In this sub-section, we compare the quality of the all-pairs shortest path lengths created using our approximation algorithm to those created by Floyd-Warshall's algorithm.

Tables 4 and 5 give the total lengths of the all-pairs shortest paths for 4 differently sized sets of data for random graphs. For each data size that is being tested, the density of graphs vary, ranging from sparse (p = 0.3) to dense (p = 0.8).

Tables 6 and 7 give the total lengths of the all-pairs shortest paths for 4 differently sized sets of data for geometric graphs. For each data size, graphs of increasing density (represented by increasing number of expected number of adjacent edges) are being tested.

Probability of Edge Existence	Total Length Error (%)	Length Error Per Edge (%)	Generation Time Saved (S)
0.3	48.37	0.0048	25
0.5	28.05	0.0028	33
0.8	8.05	0.0008	31

Table 4 Random Graph, 100 Points

Probability of Edge Existence	Total Length Error (%)	Length Error Per Edge (%)	Generation Time Saved (S)
0.3	49.85	0.0012	220
0.5	18.86	0.0005	279
0.8	7.10	0.0002	201

Table 5 Random Graph, 200 Points

Expected # of Adjacent Edges	Total Length Error (%)	Length Error Per Edge (%)	Generation Time Saved (S)
7	46.70	0.00467	22
10	19.32	0.00193	22
20	20.92	0.00209	23
50	12.95	0.00130	23

Table 6 Geometric Graph, 100 Points

Expected # of Adjacent Edges	Total Length Error (%)	Length Error Per Edge (%)	Generation Time Saved (S)
10	33.03	0.00083	200
20	24.94	0.00062	228
50	25.28	0.00063	227
100	19.77	0.00049	215

Table 7 Geometric Graph, 200 Points

Note that the Total Length Error % is given by the expression below:

$$\frac{\text{(Approx algo total path length - Floyd-Warshall algo total path length)}}{\text{Floyd-Warshall algo total path length}} \; 100\%$$

$$\text{\% of Length Error per Edge} = \frac{\text{Total Length Error \%}}{(\text{ Total no of points in the graph})^2}$$

Time saved = Floyd-Warshall Time – Approximation Algo Time

4.2.3 Observations

- The length error percentage per edge gets smaller as the problem size increases. This shows that the rate of error increase with the problem size is very slow.

- As the problem size increases, the gain in computation time of our approximation algorithm over Floyd-Warshall's algorithm increases.

- We observed that for most sizes of test problems, the total length error percentage decreases as the density of the graphs increase. This is because a denser graph means there will be more direct edges present between the points. Thus, a denser graph is more immune to the mistakes made by the approximation algorithm, since most shortest paths are the direct edges which lengths are already available.

- Our approximation algorithm has produced a good estimate of all-pairs shortest paths, incurring an error of < 0.03% per edge for random graphs and 0.01% per edge for geometric graphs.

- Finally, a plausible reason why our approximation algorithm gives better estimates for geometric graphs than for random graphs may be due to the lower probability of the occurrence of bad clustering.

4.3 Tour Length and Time Analysis

Most TSP heuristics assume an underlying complete graph connecting points. To simulate the complete graph model in a sparse graph, we need to compute the all-pairs shortest paths between all pairs of points. However, using Floyd-Warshall's algorithm for such computation is very time-consuming. Since not all pairs of shortest paths will be used in the final construction of the shortest tour, our approximation algorithm can be used to provide a quick approximation of the shortest paths between all pairs of points. Furthermore, our approximation algorithm has shown to give a good estimation of most shortest paths and it should more than sufficiently be able to produce the necessary information for the TSP heuristic to generate a reasonably good tour.

After the order of the visitation of points is decided by the TSP heuristic, we can then use Dijkstra's algorithm to generate the actual tour, ie use Dijkstra's algorithm to compute the actual shortest path and intermediate points between consecutive pairs of points in the tour. Since the order of the visitation of points is designed to form a short tour, we can safely assume that most inter-point paths are reasonably short. Thus, once the order of the visitation of points is known, the computation time of an actual tour length will be very short as well.

Next, we study the quality of the tours constructed by the various heuristics when they are given the approximate and actual all-pairs shortest paths information. We tested problems of different sizes and densities on the various TSP heuristics. Tables 8 to 11 are summaries of our experimental results. This experiment involved two types of graphs, namely geometric and random. The heuristics implemented include the Nearest Neighbour (NN), the Space-Filling Curve (SP), NN with 2-Opt, and partitioning using SP in conjunction with the NN-2Opt for post optimization (see Foo (1996) [5]).

	Tour Length Difference (%)			
Heuristic Name	NN	SP	NN-2Opt	Partition-SP-NN-2Opt
Probability				
0.3	2.32	0	3.84	5.79
0.5	3.26	0	1.86	8.09
0.8	6.60	0	0.84	2.43

Table 8 Tours Generated from a Random Graph with 100 Points

	Tour Length Difference (%)			
Heuristic Name	NN	SP	NN-2Opt	Partition-SP-NN-2Opt
Probability				
0.3	-1.34	0	2.66	4.92
0.5	-2.31	0	-2.90	1.31
0.8	3.42	0	0.38	5.63

Table 9 Tours Generated from a Random Graph with 200 Points

	Tour Length Difference (%)			
Heuristic Name	NN	SP	NN-2Opt	Partition-SP-NN-2Opt
Expected # of Adjacent Edges				
7	-12.71	0	1.43	-1.50
10	-4.25	0	-2.77	1.79
20	0	0	0.32	3.16
50	0.59	0	-1.37	0

Table 10 Geometric Graphs with 100 Points

	Tour Length Difference (%)			
Heuristic Name	NN	SP	NN-2Opt	Partition-SP-NN-2Opt
Expected # of Adjacent Edges				
10	-0.02	0	0.60	1.62
20	-10.15	0	0.08	-0.68
50	-7.33	0	-0.12	2.77
100	-7.09	0	-1.81	0.52

Table 11 Geometric Graphs with 200 Points

From the above tables, we note that even when our approximation algorithm makes mistakes in the approximation of shortest paths especially for sparse graphs, the corresponding tour length difference remains quite small. This is because the errors , made by the approximation algorithm in estimating shortest path distances usually involve points which are far apart. Since all TSP heuristics seek to minimize the final tour length, most of the path lengths between points are short. Thus, the wrong estimation of shortest path lengths will not affect the final tour construction too much.

Based on our results, we observed that better tours can be generated using estimated shortest paths information for geometric graphs than for random graphs. This may be due to the better approximation information generated for geometric graphs than for random graphs.

5 A PC Vehicle Routing Application

Based on the algorithms we have studied in the earlier sections, we have built a PC Vehicle Routing Application using Singapore's road network. Our road network comprises of all major roads and minor roads in Singapore. The total number of vertices (junctions) in our graph is about 16000. A screen shot of the system is given in Figures 5 (a) and (b) below:

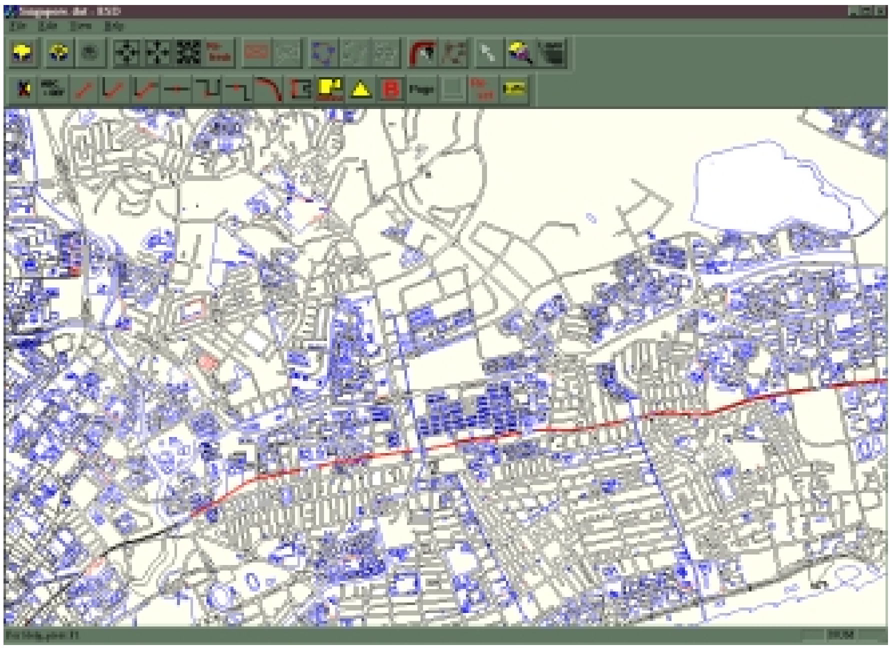

Figure 5 (a) Zooming In

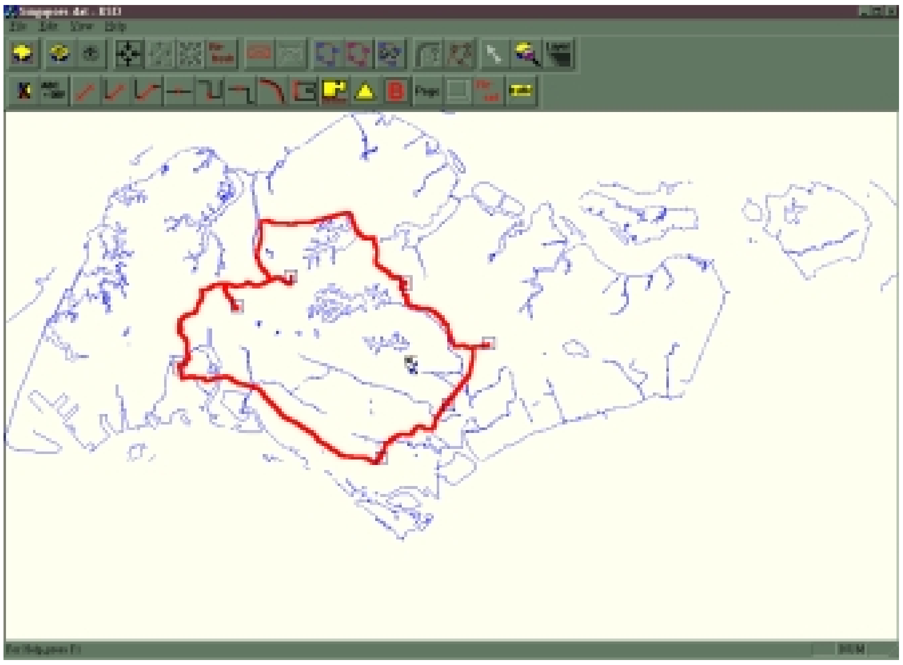

Figure 5 (b) TSP routing in the application

6 Conclusion

In this paper, we experimented with various well-known traveling salesman heuristics and proposed a hybrid that can trade off the quality of solution with computing time. The input to any TSP heuristic is the all-pairs shortest paths of the points to be visited. The traditional method of using Floyd-Warshall's Algorithm to compute all-pairs shortest paths is too time-consuming, especially for large data sets. Hence, we proposed an algorithm that generates the approximate all-pairs shortest paths. Our experimental results showed our algorithm's error to be quite small. More importantly, these estimated shortest paths information are sufficient for TSP heuristics to produce good tours especially for geometric graphs such as road networks. Our approximate all-pairs shortest paths also allow pre-computation which can speed up the shortest path computation tremendously. All these algorithms are implemented and integrated into a vehicle routing application that can be run on a PC.

References

[1] Bodin, L.D. and B.L. Golden (1981), "Classification in Vehicle Routing and Scheduling", Networks, 11, 97-106.

[2] Golden, B.L. and A.A.Assad (1988), "Vehicle Routing : Methods and Studies", North Holland, Amsterdam.

[3] Jon Bently (1992), "Fast Algorithms For Geometric Traveling Salesman Problems", ORSA Journal On Computing, 4(4), 387-410.

[4] Gerhard Reinelt (1994), "The Traveling Salesman – Computational Solution for TSP Applications", Springler-Verlag.

[5] Ching-Peng Foo (1996), "Vehicle Routing System", Honours Thesis, Department of Information Systems and Computer Science, National University of Singapore.