Automatic Tour Plan Generation for Mobile Robots

by
J. Borenstein, Assistant Research Scientist
and
Y. Koren, Professor

Technical Report No. UM-MEAM-89-03
April 1989

ABSTRACT

This paper introduces a Tour Plan Generator for mobile robots. Tour generation for mobile robots is a special case of what is known as a "Vehicle Scheduling Problem". In this paper, the particular requirements for mobile robots are identified, and a Tour Plan Generator is designed which address these requirements.

This TPG combines different heuristic tour-construction rules into a "*heuristic team approach*". A new heuristic rule has been tested and was found to yield superior results when part of the *heuristic team*. Finally, an Index of Performance has been defined which allows to evaluate the performance of individual members of the *heuristic team*.

engn
UMR Ø561

1. INTRODUCTION

A task-oriented navigation system for mobile robots is currently under development in the Robotic Systems Division at the University of Michigan. This hierarchically organized system comprises four functional levels of performance.

The *Task Planner* (TP) represents the system's highest level (Level IV). Interfacing with the human operator, the TP receives a list of tasks that need to be performed by the robot. Out of this list, the TP extracts all locations pertinent to mobile robot travel and sends them to the next lower level.

Level III is the *Tour Plan Generator* (TPG). The TPG plans a sequence of *trips* among all specified locations. This is not a trivial task, since some locations have to be visited in a certain order, and some potential tours may be to long and have to broken up into smaller ones.

Level II of the system is the *Global Path Planner* (GPP). The GPP contains (or has access to) a world model which includes information about stationary obstacles (e.g., walls), off-limit zones (e.g., stairs), and information about recently detected unexpected obstacles. Based on this information, the GPP plans an optimal path between the current location and a goal location (Borenstein and Koren, 1986). The path produced by the GPP is expressed as a linked list of via-points, typically spaced about 1 to 10 meters apart.

At the lowest level (Level I), a *Local Path Planner* (LPP) drives the mobile robot from one via-point to the next. The main task of the LPP, however, is obstacle avoidance (Borenstein and Koren, 1888, 1989a, 1989b). In our mobile robot system, the LPP uses ultrasonic sensors to detect and avoid obstacle on the fly, while continuing to proceed toward the designated target location.

This paper deals with some aspects of Level III, the *Tour Plan Generator*. The heart of the TPG is an algorithm which computes an optimal (or near-optimal) tour. A tour comprises of one start location and any number of intermediate-goal locations. It is assumed that a tour is closed, that is, the robot returns to its original start location.

This type of problem falls into the class of "Vehicle Scheduling Problems" and is of interest to a variety of transport applications. However, mobile robots represent a special class of transportation device and therefore need special considerations.

The TPG described in this paper is especially designed for mobile robot applications and offers the following features:

1.1 Distribution

This is the most basic feature of the TPG (in this form, it is also known as the "*Traveling Salesman Problem*".) A *distribution* task requires the mobile robot to get a supply of items at a central depot, and then distribute these items to a number of drop-off points. For this purpose, the TPG will calculate an optimal tour (in terms of distance), starting at the depot and passing each specified drop-off point exactly once, before returning to the depot. Usually, the order in which drop-off points are visited is unimportant.

In this basic case, the robot's transport capacity is considered unlimited, that is, it is assumed that there is always enough capacity to load all distribution items for a tour.

1.2 Tour Length Constraint

A mobile robot's maximal tour length may be limited to a certain "mileage" between necessary visits to a "home-depot". Typical reasons for this requirement may be the need to recharge batteries, or to recalibrate an onboard position system (e.g., a gyro).

The TPG inserts "home-depot visits" into the tour, after a preprogrammed tour-length has been traveled. The overall tour-length, comprising all such created *subtours*, is again optimal. While not expressively addressed in this paper, this feature can easily be modified to deal with load-capacity limitations.

1.3 Ordered Pairs

This feature addresses situations in which only one item can be transported at a time. This situation is characteristic for mobile robots equipped with an arm. Whenever the arm is used to carry an item, it is only this one item that can be held in the robot's gripper. Consequently, goal locations are not treated equally, but rather as *Ordered Pairs*, where each source location (where an object is acquired), must be followed immediately by its respective goal location (where the object is released) before the robot can perform the next task.

1.4 Mixed Tasks

With the described TPG, *Distribution* and *Ordered Pair* Tasks can be handled concurrently. This means that the operator may issue a list of mixed commands (command keywords are printed in upper-case), such as:

"BRING item_1 FROM place_1 TO place_2 AND DISTRIBUTE item_2 to place_3 AND DISTRIBUTE item_3 to place_4 AND MOVE TO place_5 AND BRING item_4 (robot knows where) TO place_6 AND [more activities] AND [more activities] PLEASE".

The TPG computes a sequence of *trips* which connect all locations, such that the overall traveling cost (in terms of distance) is optimal.

2. THE TRAVELING SALESMAN PROBLEM

Features 1.1 through 1.4 above are based on a well known problem in the fields of operations research and artificial intelligence, which is known as the "*Traveling Salesman Problem*" (TSP). The problem is to find an optimal tour through *n* cities, starting at city 1, then visiting the *n*-1 remaining cities once and only once before returning to the origin. Optimallity is normally defined in terms of minimum distance or minimum cost.

The problem is of great interest because of its many "real-life" applications, among which are fuel oil delivery, newspaper distribution, or delivery of any kind of goods from a central depot to a number of outlets. One approach to solve the problem is based on branch and bound algorithms (also known as A^*) (Nicholson, 1967; Miliotis, 1976), which yield exact (optimal) solutions if the number of cities involved is small. Unfortunately, the combinatorical nature of the problem tends to increase computer-storage and calculation-time requirements dramatically with increasing *n*. As a matter of fact, the problem has been shown to belong to the class of NP-complete problems, which require an amount of time exponential in *n* for an exact solution.

Therefore, much effort has been put into the development of heuristic algorithms, which, by their very nature, are not necessarily optimal, but may be evaluated in statistical terms. This is justified by the fact that a "good" solution is acceptable for most practical applications. In addition, almost-optimal algorithms have been tested that average at only 1 percent cost above the known optimal solution's cost (Golden et al., 1980) for fairly large problems ($n=100$). For this reason, we will use the term "optimal" in the sense of "almost-optimal" or "the-best-solution-the-heuristic-algorithm-could-produce". By contrast, when refering to the mathematically exact optimal solution, we will use the term "*absolute optimal*".

The heuristic solutions for the TSP fall into three classes:

a) **Tour construction procedures**
where the heuristics are applied during construction of the tour, by proper choice of the next city to be included and its location in the growing *subtour*. Examples for this methods are:

- Nearest Neighbor, discussed and analyzed in (Rosenkranz et al., 1974).

- Clarke and Wright Savings, discussed and improved in (Golden, 1977a).
- Nearest-, arbitrary-, or farthest insertion, analyzed and tested in (Rosenkranz et al., 1974).
- Cheapest Insertion, introduced in (Raymond, 1969).
- Convex Hull (Wiorkowski and McElvain, 1975; Golden et al., 1980).

b) **Tour improvement procedures**
where a randomly constructed initial tour is improved by exchanging one subset of links in the tour with another subset, until no more improvement is achieved. Examples for this method are:

- inversions or 2-opt method, introduced in (Croes, 1958).
- 3-opt method, introduced in (Lin, 1965).
- k-opt method, introduced in (Lin and Kernighan, 1973).

c) **Composite procedures.**
This is a simple yet effective refinement: Firstly, a tour construction procedure is applied in order to produce a "good" initial tour. Secondly, one or even two tour improvement procedures are applied. This method, in several variations, has been tested and compared in (Golden et al, 1980) and a FORTRAN program is given in (Zdenek, 1973).

A thorough comparison between all the above methods, with many experimental results (concerning accuracy and timing) is given in (Golden et al, 1980). A theoretical analysis of some of the tour constructing procedures is given in (Rosenkranz et al, 1974).

2.1 Limitations and Assumptions

There are two assumptions for the TSP in its basic form:

2.1.1 Symmetrical Distance Matrix

This assumption applies for the common case where the distance between city *i* and city *j* is the same in both directions). An example for a case which produces asymmetrical distances are intraurban connections, with one way streets, where the actually traveled distance between two locations is not the same in both directions.

2.2.2 Triangle Inequality

The triangle inequality is fundamental in euclidean geometry, stating that the sum of the lengths of any two sides in a triangle is greater than the length of the third side. The TSP, applied to interurban connections in hilly areas, where roads may be much longer than the direct distance between two adjoining cities, is an example for a case were the triangle inequality is not necessarily satisfied.

Not all of the above algorithms are capable of handling exceptions from the basic form of the TSP. Unfortunately, experimental results in (Golden et al, 1980) relate only to the basic form of the TSP, and so do most other papers on the subject.

2.2 Vehicle Routing

A related class of problems is known as the *Vehicle Routing Problem* (VRP). In the VRP, one must design a set of routes of minimal total cost, leaving from and eventually returning to a depot, while satisfying capacity constraints and meeting customer requirements (Federgruen and Zipkin, 1984). The problem can be seen as a (at times much more demanding) extension of the TSP, adding some or all of the following requirements and restrictions beyond the standard TSP:

- An optimal tour between one central depot and many customers has to be found for many vehicles traveling concurrently on many *subtours*.
- Vehicles have maximum capacities.
- Vehicles have maximum route time constraints.
- Multiple depots, multiple capacities, multiple demands, etc.

The approach in (Federgruen and Zipkin, 1984) and (Golden et al, 1977) is based on reducing the complexity of the VRP (by means of some heuristics), to the basic TSP, which may then be solved by any of the methods discussed above. Clearly, this approach yields only an approximate solution. Moreover, it is very difficult to evaluate the performance (in terms of accuracy) of the suggested algorithm, since exact solutions are known only for very small problems.

3. A TOUR PLAN GENERATOR FOR MOBILE ROBOTS

We have developed two models for mobile robot applications which allow to implement our TPG as a special case of the *Traveling Salesman Problem*.

Model I:
The foremost problem in tour planning for mobile robot applications stems from the fact that many mobile robots (especially when carrying an object in their gripper) have a limited transport capacity, namely, one item at a time. In the case of the *Ordered Pair* feature, where the robot is required to transport several objects (one at a time) from *source* locations to *goal* locations (*source* and *goal*, respectively, in the following), the TSP can not readily be employed. This is so, because in this problem a *source* must always be visited immediately prior to visiting a *goal* — a constraint the basic TSP is not equipped to handle.

As a solution to this problem, we suggested that each *ordered pair* (*source* and *goal*) are treated as **one** location in the TSP. This concept can be visualized by picturing the vehicle entering a location *i* at *source(i)* and emerging from that location at *goal(i)* (see Fig. 1). The cost matrix $C(n,n)$ for this model is asymmetric, with elements *c(i,j)* being distances between *goal(i)* and *source(j)*. As is seen in Fig. 1, the distance between *goal(i)* and *source(j)* differs from the distance between *source(i)* and *goal(j)*.

The diagonal elements in C represent the distance between *source* and *goal* of the same *Ordered Pair*. Since the vehicle must travel from each *source* to its respective *goal*, anyway, diagonal elements are not considered in the algorithm at all, and may be set to 0 (they are considered, nevertheless, but for another purpose, which will be explained later).

It should be noted that tasks involving only one physical location at a time (e.g., *Distribution* tasks) can easily be adapted to the above concept by simply assigning the same coordinates to artificially defined *sources* and *goals*. This is why *Ordered Pair* and *Distribution* task commands can be issued in the same command sequence.

From Fig. 1 one can see that the triangle inequality does not hold true in this case: the triangle tour 1' to 2; 2' to 3; 3' to 1, constituting a complete tour according to our model, does not yield $c(1'\text{-}2) + c(2'\text{-}3) > c(3'\text{-}1)$.

In practical mobile robot applications the triangle inequality does not hold for yet another reason: a distance in the cost matrix is actually the length of a *trip*, which is computed as the shortest possible connection between two locations, **under avoidance of known obstacles.** Therefore a *trip* may be considerable longer than the straight (euclidean) distances between two locations.

As is seen from the above, the requirements in mobile robot applications do not meet any of the two assumptions of the basic TSP. Although some of the TSP-algorithms are still applicable, most of the experimental and analytical results from literature are not, since they assume symmetric cost matrices and euclidean distances.

Model II:

In order to insert a "home-depot visit", after a preprogrammed tour length L_{max} has been traveled, the tour must be broken up into *subtours*, each of which starting and terminating at the "home"-location. The length of the *subtour* is not to exceed L_{max}. Once a suitable mechanism for breaking up the complete tour into *subtours* is devised, it is easy to alter the physical attribute producing the cost which is not to exceed L_{max}. Battery charge, for example, in a battery powered vehicle; time, in school-bus scheduling or newspaper dispatching; capacity, in a vehicle with capacity limitations; or position accuracy of a position sensor (e.g., a gyro) which drifts with increasing tour length or time.

4. THE TPG ALGORITHM

Out of the three classes of heuristic TSP algorithms (see Section 2), we have chosen to work with *tour construction procedure*, since these offer a straight-forward way to accommodate Model II. A *tour construction procedure* (also called "insertion procedure") usually starts with the smallest possible *subtour* (2 locations), inserting new locations to this *subtour* according to a heuristic rule, until all locations are included and the tour is completed. At each stage, after a new location has been inserted, the *tour length constraint* can be tested. This is done by comparing the accumulated tour-length (of the current *subtour*) to L_{max}. If L_{max} is exceeded, the last inserted location is removed and the remaining *subtour* (now complying with L_{max}) is saved. Subsequently, a new *subtour* is constructed, until all locations have been included. Note that the a *subtour* is optimal at any stage of this process, since the newly inserted location was selected by the algorithm's heuristic selection rule.

The following is the generic form for the "closest" (or "farthest", or "arbitrary") insertion algorithm for the standard TSP:

1. choose a starting location *i*.
2. choose a location *k* such that $c(i,k)$ is minimal (or maximal) and form *subtour i-k-i*.
3. find location *k* not yet in the *subtour*, which is closest (or farthest, or arbitrary) to any location in the *subtour*.
4. find *trip* (i,j) in the *subtour* which minimizes $c(i,k)+c(k,j)-c(i,j)$. Insert *k* between *i* and *j*.
5. repeat 3. and 4. until all locations are included in the tour.

Calculation times for such algorithms are of the order of n^2 (Golden, 1980). Commonly, one would run the same algorithm *n* times, choosing each one of the *n* locations once as the starting location, comparing costs of each run and selecting the lowest cost as the result. This changes the order of the resulting algorithm to $O(n^3)$, but considerably improves the quality of the solution.

In our application, however, *subtour* are likely to be created because of the *tour length constrain*. Since *subtours* have to start and end with the same location, one can not run the algorithm with different starting locations, rendering this means of improvement useless.

Nevertheless, we were able to improve results by running **several similar** algorithms for the same problem, again selecting the best run to yield the resulting tour. We run the generic insertion algorithm (comprising steps 1 through 5, above), with variations to the *selection rule* (step 3). The following six *selection rules* were employed:

SELECT1: Find location k, not in *subtour*, farthest from starting location

SELECT2: Find location k, not in *subtour*, closest to last chosen location

SELECT3: Find location k, not in *subtour*, close to last chosen location, but far from 1st location, minimizing $c(1,k)-c(k-1,k)$

SELECT4: CLOSEST INSERTION: find location k, not in *subtour*, closest to any location in *subtour*

SELECT5: FARTHEST INSERTION: find location k, not in *subtour*, farthest from any location in *subtour*

SELECT6: CHEAPEST INSERTION: for (i,j) in *subtour* find location k, not in *subtour*, minimizing $c(i',k) + c(k',j) - c(i',j)$.

5. EXPERIMENTAL RESULTS

In order to obtain statistically relevant data, 50 random problems, each comprising $n=15$ locations (1 starting location and 7 *start-goal* pairs), were created. Table 1 gives the coordinates for one such problem and Table 2 shows the associated asymmetric cost matrix C. The diagonal elements of C hold the distance between the *start* and *goal* location of an *Ordered Pair*. This information is not needed (and could be set equal 0) for choosing an optimal tour, since this distance has to be traveled anyway, for any possible tour. However, this distance must be considered on behalf of the *tour length constraint*.

RUN#32				
	COORDINATES			
#	X'	Y'	X	Y
1	718	279	718	279
2	507	312	705	171
3	33	171	620	233
4	201	11	561	164
5	650	258	391	279
6	595	343	297	330
7	186	143	373	327
8	532	170	479	339

Table 1: Example-run coordinates

RUN#32								
		ASYMMETRIC COST-MATRIX C						
#	1	2	3	4	5	6	7	8
1'	0	108	108	194	327	424	348	246
2'	213	243	137	157	120	210	134	38
3'	693	672	590	528	373	308	374	476
4'	582	528	474	391	328	333	359	429
5'	71	102	39	129	259	360	285	189
6'	138	204	112	182	213	298	222	116
7'	549	519	443	375	246	217	262	352
8'	215	173	108	29	178	284	223	177

Table 2: Asymmetric cost-matrix for example run.

Fig. 2 depicts the same example graphically. The starting location is labeled "1", *start/goal* pairs are labeled "2" through "8", and *goal* locations are distinguished by apostrophes. Close inspection of Fig. 2 shows the difficulty in finding an optimal tour manually, even for this relatively small problem.

For each one of the 50 problems, the *absolute* optimal solution was calculated by comparing the costs for all possible tours. There are $[(n-1)/2]!$ (=5040, for $n=15$) possible permutations for each of the problems, which is why only "small" problems were created in our experiments.

In order to enforce the creation of a tour comprising at least two *subtours*, L_{max} was then set to 75% (arbitrarily chosen value) of the *absolute optimal* cost. (Clearly, if the admissible maximal length of a *subtour* (L_{max}) is less than the known *absolute optimal* solution, then the heuristic algorithm will have to build two or more subtours.) Recalculation of the new optimal cost, now under consideration of the maximal cost constraint (L_{max}), yielded a new optimal tour, as illustrated for the above example, in Table 3. The value of C_{opt} in Table 3 represents the total cost of the *absolute optimal* solution of the 2-*subtour* problem created through the *tour length constraint.*

```
RUN#32   SEED WAS:-21848   5040 tours created.
Absolute optimal tour (with 1 subtour): 1 2 8 4 7 5 3 6 1
Copt = 3485 (with one subtour)
Lmax = 2613 (set to 0.75*Copt to force 2nd subtour)
Absolute optimal tour (with 2 subtours): 1 3 6 1 2 8 4 7 5 1
Copt = 3625   (with 2 subtours)
```

Table 3: Absolute optimal tour with one and two subtours.

Subsequently, each one of the six heuristic algorithms was run for each of the 50 random problems. A typical result (for the above example) is shown in Table 4, where the relative error E_{rel} (as compared to to C_{opt}) of each *selection rule* is shown. Since the TPG always runs all of the six *selection rules* for each problem, The TPG can chose the best result out of the six and make it the "representative" result. In the example run of Table 4, *selection rule* #3 happened to find the exactly optimal tour, and therefore the *representative result* shows $E_{rel}=0$. Obviously, this is coincidental, and *representative results* have mostly $E_{rel}>0$. A better indication would be an *average* of the *relative errors* of *representative results* over a large number of problems. We will call such an average E_{avrg}.

RUN#32 SEED WAS:-21848 L_{max}=2613 C_{opt}=3625

Selection Rule	Tour	Cost	E_{rel}
SELECT1	1 3 7 6 5 1 2 4 8 1	4112	13.43%
SELECT2	1 8 4 7 6 5 1 2 3 1	4293	18.42%
SELECT3	1 2 8 4 7 5 1 3 6 1	3625	0.00%
SELECT4	1 2 8 3 7 5 1 4 6 1	3830	5.65%
SELECT5	1 2 8 3 7 6 1 4 5 1	3796	4.71%
SELECT6	1 7 6 8 4 5 1 2 3 1	4267	17.71%
TPG's best	1 2 8 4 7 5 1 3 6 1	3625	0.00%

Table 4: Results of individual selection rules.

6. INDEX OF PERFORMANCE FOR HEURISTIC SELECTION RULES

As is evident from Table 4, the various *selection rules* performed quite differently on the same problem. However, none of the *selection rules* performed always well or always poorly on all 50 problems.

One can easily think of additional heuristic *selection rules* and add them to the TPG. Even if the rules were bad, they may produce the best tour, every once in a while. However, with more *selection rules* to run, computation time for the TPG grows. A good evaluation method would allow the programmer to include only those heuristic rules that contribute significantly. The question therefore arises, how to evaluate the performance for heuristic *selection rules*. One approach to this question would be to calculate the average *relative error* produced by each *selection rule* when run alone. A result of this test is shown in Table 5.

Active rule	Average error E_{avrg} (50 runs)
only SELECT1	7.22%
only SELECT2	7.12%
only SELECT3	8.00%
only SELECT4	7.63%
only SELECT5	7.45%
only SELECT6	10.03%

Table 5: Average error E_{avrg} produced by each selection rule, when running alone.

However, since in the actual algorithm the rules function as a "team", their performance should be evaluated in that context. For this reason, we introduced another test. This time, the TPG was run for all 50 problems, but each time *omitting* one of the six selection rules (i.e., the *representative result* was chosen out of the remaining five rules.) The better C_{avrg} resulting from the remaining *selection rules*, the lesser the "would-be" *contribution* of the omitted *selection rule*. To illustrate this approach, one can imagine a creative "think-tank" team, working jointly on a problem: If only the best idea counts, it is less important how often a team-member comes up with *another good* idea, but rather how often he or she produced *the best* one.

Table 6 shows E_{avrg} for the TPG using all but one *selection rule*. The first entry, with all *selection rules* active, produced an average cost of 2.39% above the optimal cost. In each of the following runs, one of the *selection rules* was omitted.

As can be learned from Table 6, by far the best *contribution* comes from *selection rule 3*. (To our knowledge, the heuristics of this rule have not been suggested elsewhere in the context of the TSP. However, it must be emphasized that this experiment has been performed with a *tour length constraint*). Intuitively, *selection rule 3* can be understood as an attempt to fill as many remote (yet close to each other) locations as possible into one *subtour*, thereby avoiding high costs in cases where a *subtour* had to be closed due to L_{max} before covering all locations in a remote area.

Active selection rules	Average error E_{avrg} (50 runs)	Contribution to joint performance
All Selection rules	2.39%	
all except SELECT1	2.69%	0.30 = 11.15%
all except SELECT2	2.52%	0.13 = 5.16%
all except SELECT3	3.07%	0.68 = 22.15%
all except SELECT4	2.76%	0.37 = 13.41%
all except SELECT6	2.51%	0.12 = 4.78%

Table 6: Average error E_{avrg} produced by all (or all except one) selection rules.

7. CONCLUSIONS

A Tour Plan Generator (TPG), optimized for the specific needs of mobile robot applications, has been developed. Features of this TPG include the ability to automatically insert home-depot visits into the tour, as well as to deal with ordered *source/goal* pairs which have to be visited in proper sequence. A model has been developed which allows to represent the latter problem as a TSP with asymmetric cost matrix and non-euclidean distances.

New heuristic tour construction procedures, designed specifically for mobile robot requirements have been tested and found to compare favorably with known heuristics.

A "*heuristics team approach*" has been employed to further improve the TPG's performance, and an original way to evaluate the performance of individual components (selection rules) of the heuristic team has been introduced.

With the index of performance method described in Section 6, an efficient tool has been introduced in order to accurately evaluate the performance of additional *selection rules*. This method is particularly suitable for the *heuristic team approach* used in this TPG.

8. REFERENCES

Borenstein, J. and Koren, Y., 1986 "Optimal Path Algorithms For Autonomous Vehicles". Paper presented at the *13th CIRP Manufacturing Systems Seminar*, June 5-6, Stuttgart, West-Germany.

Borenstein, J. and Koren, Y., 1988, "High-speed Obstacle Avoidance for Mobile Robots". *Proceedings of the IEEE Symposium on Intelligent Control*, Arlington, Virginia, August 24-26.

Borenstein, J. and Koren, Y., 1989a, "Real-time Obstacle Avoidance for Fast Mobile Robots". Accepted for publication in the *IEEE Trans. on Systems, Man, and Cybernetics*.

Borenstein, J. and Koren, Y., 1989b, "Real-time Obstacle Avoidance for Fast Autonomous and Semi-autonomous Mobile Robots". *Third Topical Meeting on Robotics and Remote Systems*, Charleston, South Carolina, March 13-16, 1989.

Croes, G. A.: "A Method For Solving Traveling-Salesman Problems." *Operations Research 5*, 1958, pp. 791-812.

Federgruen, A., and Zipkin, P.: "A Combined Vehicle Routing and Inventory Allocation Problem". *Operations Research*, Vol. 32, No. 5, Sept.-Oct. 1984, pp. 1019-1037.

Golden, B. L., 1977, "Evaluating a Sequential Vehicle Routing Algorithm". *AIIE TRANSACTIONS*, Vol. 9, No 2, pp. 204-208.

Golden, B. L., Magnanti, T. L., and Nguyen, H. Q.: "Implementing Vehicle Routing Algorithms". *Networks*, Vol. 7, 1977, pp. 113-148.

Golden, B., Bodin, L., Doyle, T., and Stewart, W. Jr.: "Approximate Traveling Salesman Algorithms". *Operations Research*, Vol. 26, No. 3, May-June 1980, pp. 694-711.

Lin, S. and Kernighan, B. W.: "An Effective Heuristic Algorithm for the Traveling-Salesman Problem". *Operations Research 21*, 1973, pp. 498-516.

Lin, S.: "Computer Solutions of the Travelling Salesman Problem". *Bell System Tech. Journal*, Vol. XLIV, No. 10, December 1965.

Miliotis, P.: "Integer Programming Approaches to the Traveling Salesman Problem". *Mathematical Programming*, Vol. 10, 1976, pp. 367-378.

Nicholson, T. A. J.: "A Sequential Method for Discrete Optimization Problems and its Application to the Assignment, Traveling Salesman, and Three Machine Scheduling Problems". *J. Inst. Maths Applics*, Vol. 3, 1967, pp. 362-375.

Raymond, T. C.: "Heuristic Algorithm for the Traveling Salesman Problem". *IBM Journal of Research and Development*, July 1969, pp. 401-407.

Rosenkrantz, . J., Stearns, R. E., and Lewis, P. M.: "Approximate Algorithms for the Traveling Salesperson Problem". *Proceedings of the 15th Annual IEEE Symposium of Switching and Automata Theory*, 1974, pp. 33-42.

Wiorkowski, J. J. and McElvain, K.: "A Rapid Heuristic Algorithm for the Approximate Solution of the Traveling Salesman Problem". *Transportation Research*, Vol. 9, 1975, pp. 181-185.

Zdenek, F.: "Algorithm 456. Routing Problem [H]". *Communications of the ACM*, Vol. 16, No. 9, Sept. 1973, pp. 572-574.

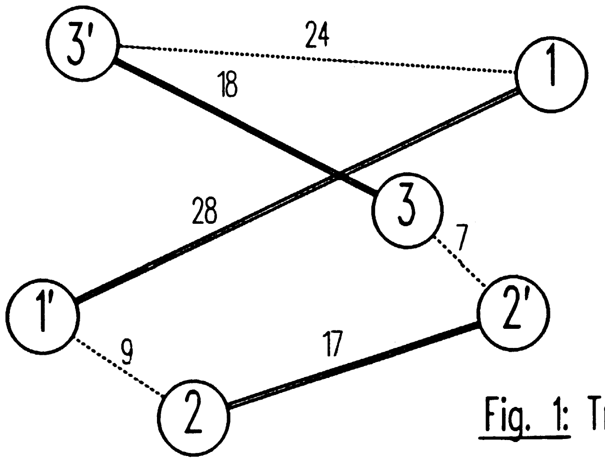

Fig. 1: Triangle Inequality does not hold when using the ordered pair model.

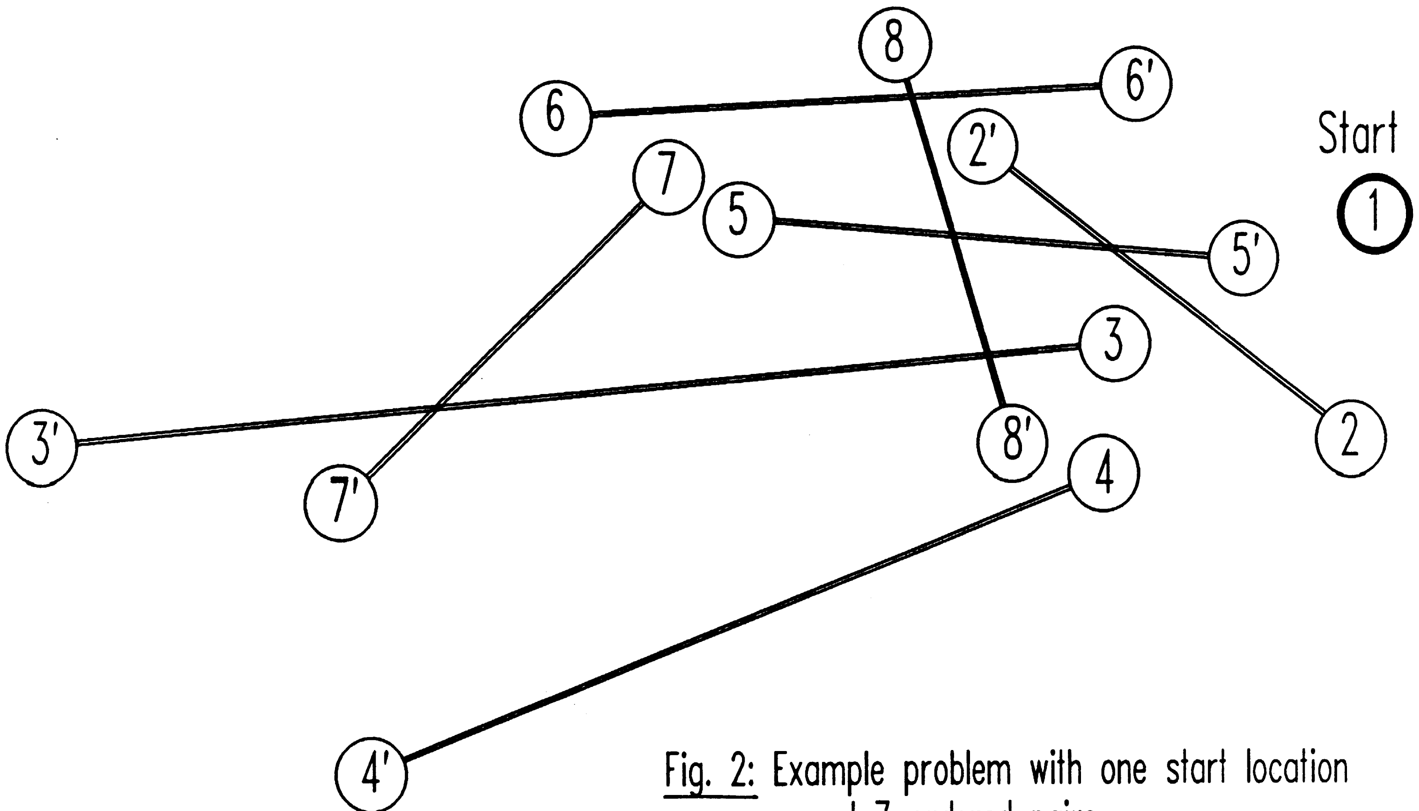

Fig. 2: Example problem with one start location and 7 ordered pairs.