To the Graduate Council:

I am submitting herewith a thesis written by Melinda Vazzana entitled "A GIS Implementation of a Time-dependent Traveling Salesman Problem with Stochastic Events." I have examined the final paper copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Geography.

Bruce Ralston, Major Professor

We have read this thesis
and recommend its acceptance:

Shih-Lung Shaw

Cheng Liu

Accepted for the Council:

Vice Provost and Dean of Graduate Studies

Thesis
2003
.V29

A GIS Implementation of a Time-dependent Traveling Salesman Problem with Stochastic Events

A Thesis
Presented for the
Master of Science
Degree
The University of Tennessee, Knoxville

Melinda Vazzana
August 2003

Acknowledgements

I would like to dedicate this thesis to my beloved husband and best friend, Geoffrey H. Frost. You have been my sanctuary when it all became too much for me and have amazed me with your editing skills and grasp of grammar and syntax. I owe you so much for your patience and support this past two years and I want to thank you for making this work possible.

I would also like to thank my mother who instilled in me my drive to not be satisfied until I have given 200%, which has helped me in my two years at the University of Tennessee, Knoxville. I would also like to thank Dr. Shaw for steering me in my quest for knowledge of trasportation problems and for helping me define my thesis topic. I would like to thank both Dr. Shaw and Dr. Ralston for introducing me to the potential for customization from within ESRI's ArcInfo 8. I would also like to thank Dr. Ralston for introducing me to ESRI's COM objects of MabObjects and ArcObjects. I would also like to thank Dr. Ralston and Dr. Liu for guiding me through their recomendations on my code and implementation strategy to produce a better final solution.

Abstract

The purpose of this thesis is to provide the custom tools and functions in ArcGIS 8 to solve the transportation problem domain of the Time-dependent Traveling Salesman Problem (TDTSP) that also considers stochastic events. My implementation of a TDTSP with stochastic events takes full advantage of the open-COM based architecture of ERSI's ArcInfo 8. Through the COM-complaint programming language of C++, as well as taking advantage of the VBA editor and ArcObjects, the integration of the custom tools to solve the TDTSP is seamless to the user. Travel speeds are affected not just by time-dependent events, such as rush hour congestion, but also by stochastic events, such as traffic accidents. My study also strives to continue the applicability of the TSP model to the real world by incorporating stochastic events such as traffic accidents. To try and maintain a cheap solution, there needs to be efforts to include these stochastic events. My GIS implementation of this problem is one scenario to handle dependent and stochastic travel events.

Table of Contents

Chapter			Page
1	Introduction		1
2	Literature Review		11
	2.1	Review of Literature for the TDTSP	13
	2.2	Review of Literature for GIS and Routing	22
3	Data Description and Methodology		34
	3.1	Study Area	38
	3.2	General Approach	40
	3.3	Assumptions	46
	3.4	Time Period and Step Function	47
	3.5	Shortest Path Routine	49
	3.6	TDTSP Heuristic	53
		3.6.1 ArcInfo 8 Implementation of the TDTSP	57
		3.6.2 TDTSP Example	60
	3.7	Traffic Reports	67
	3.8	Re-Order Heuristic	70
		3.8.1 ArcInfo 8 Implementation of Re-Order	71
4	Results		73
	4.1	User Implementation of TDTSP with Stochastic Events	74
	4.2	Comparison Analysis	84
5	Conclusions and Future Research		88
References			93
Appendix			100
Vita			139

List of Tables

Table		Page
3.1	Travel speed matrix for time period 1	50
3.2	Travel speed matrix for time period 2	50
4.1	Comparison analysis results	86

List of Figures

Figure		Page
1.1	A TSP Tour	2
1.2	Visual representations of three time periods	6
2.1	A TSP tour that assumes any link *(i,j)* is during one time period	14
3.1	Example of one "pseudo-network" link	38
3.2	Map of study area	39
3.3	A TDTSP tour in progress	41
3.4	Flow chart of Steps 1 through 4	44
3.5	Flow chart of Steps 5 through 7	45
3.6	My custom ATL COM object model	55
3.7	Flow chart of the process of the TDTSP heuristic	58
3.8	Generalized view of the "pseudo-network" for time period 1	61
3.9	Generalized view of the "pseudo-network" for time period 2	61
3.10	The subtours saved in *PrevTour* for tour length 2	63
3.11	The subtours saved in *PrevTour* for tour length 3	64
3.12	The subtours saved in *PrevTour* for tour length 4	66
3.13	The final subtours saved in *PrevTour*	67
3.14	View of tools used for determining if Re-Ordering is necessary	69
4.1	My TDTSP with independent events project view	76
4.2	View of my toolbar with its custom tools and buttons	77

Figure Page

4.3 View of map after executing *BuildMatrices* 79

4.4 View of each *TourLinks#*'s table after executing *BuildMatrices* 79

4.5 View of *DisplayTour* button in action 80

4.6 View of each *TourLinks#*'s table after executing *DisplayTour* 80

4.7 View of procedure used to select affected tour links 83

4.8 View of *MyUserUpdateForm* 83

List of Plates

Plate		Page
1.1	TDTSP ATL COM Project.....	In pocket

Chapter 1

Introduction

Geographic Information Systems (GIS) specifically concerned with transportation issues and problems have been coined GIS-T. Miller and Shaw (2001) state that transportation applications are the most important implementations of GIS. Lewis (1990) noted in his article that GIS-T systems are viewed as beneficial and effective tools in reducing cost through their ability to collect, store and manage transportation data. GIS handles information, whether spatial or nonspatial, effectively through its database component. GIS goes further than a database management system (DBMS) to incorporate visualization and spatial analysis tools. Within this framework, GIS systems for transportation problem domains are increasing in demand (Miller and Shaw 2001). Thus, GIS is an excellent framework to implement a time dependent Traveling Salesman Problem (TDTSP) with independent travel events.

The Traveling Salesman Problem (TSP) is a classic routing problem that has been long studied in literature and is the core of the TDTSP. The problem scenario of the TSP is, simply stated, when a salesman starting at his home city, which I denote as 0, needs to travel to n-1 number of cities exactly once, and then return home (see Figure 1.1). The objective is to select the order of cities and the paths between them in such a way as to minimize the total travel cost. I assume that the travel cost

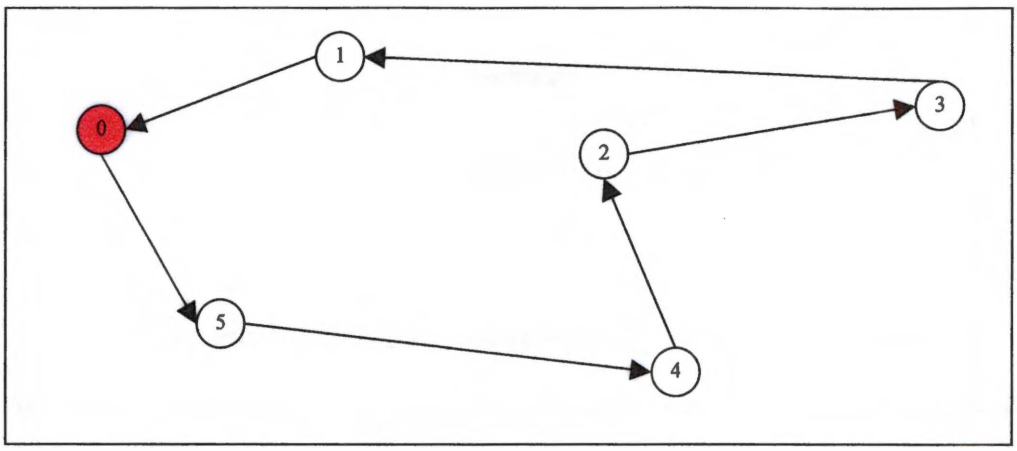


Figure 1.1 A TSP Tour.

between each pair of cities *i* and *j* is known. This is denoted as C*ij*, and is derived from the shortest path calculation between these two nodes. The visiting order of cities can be thought of as an order of *n* links of a tour. On link *k*, the salesman travels from city *i* to city *j*. The link represents the path used to travel from the origin node *i* to the destination node *j*. On link 1 he travels from the home depot to some other city, and on link *n*, he travels to city 0. Figure 1.1 gives a visual representation of a TSP tour (with the depot in red). In this example the tour follows this sequence: (0,5), (5,4), (4,2), (2,3), (3,1), (1,0).

The initial TSP problem definition has limiting assumptions that have spurred researchers to expand the TSP to include time windows, customer dynamic ordering environment, and time-dependent travel times (TDTSP), to enhance the model's ability to capture real world conditions. The business market is the driving force behind such extensions of the TSP in order to capture the business operations of time windows and dynamic ordering environment. The time-dependent extension accounts for urban networks where travel speeds are not constant functions of

distance but vary through time due to such phenomenon as congestion. These extensions to the TSP are extremely complicated problems with many interacting variables and constraints. However, such research is needed to more effectively model real world scenarios.

Time windows appear for businesses that work on fixed time schedules. Prior to adding the time window constraint to the actual algorithm formulation, the problem required this information to be added in manually by an expert user (Solomon 1987). This TSP extension has been well studied with many algorithms developed to include time window constraints. Solomon (1987) reviewed several existing heuristic algorithms concerned with Vehicle Routing and Scheduling problems (VRSP) and efforts to include time windows. The TSP is a specialized case of the VRSP with the vehicle fleet size equal to one. Solomon (1987) compared two Savings approaches (one with waiting at a stop for a customer time window to open and the other with no waiting), a time-oriented Nearest-neighbor approach, three different Insertion objective approaches and a time-oriented Sweep approach for the VRSP. These are all classified as tour construction algorithms. The Savings approach selects a node to add to a tour that results in the lowest additional cost. The Nearest-Neighbor approach selects the closest node to the last node added to the tour in terms of distance and time. The Insertion approach satisfies the objective of selecting a node not yet in the tour that can be reached at the lowest cost from any node currently in the partial tour. For Solomon (1987) these objectives are either the maximum benefit to the partial tour or minimum cost or the temporal urgency of service. The Sweep

approach decomposes the problem into a clustering and a scheduling stage. The results indicated that for solution quality the first Insertion approach of the three preposed for this research was the best. The Sweep was the best for a large number of customers per vehicle. The overall finding was that without the time windows constraint, the usefulness of the TSP model would be negligible to those businesses that function on guaranteeing the pickup/delivery of their goods/services in a specified time window.

Another aspect of the way business functions in the real world that has affected the modeling of the TSP is the dynamic environment of receiving orders. The schedule of nodes to visit might change as the day progresses with new orders coming in after drivers have left. The dynamic approach is to enter the new orders as they arrive. This method is a re-optimization procedure, since with each new order added the TSP tour has to be reformulated. Other methods in the literature include the stochastic approach, which develops a probability of demand for each customer from their set of known customers (Gendreau et al. 1995). These algorithms are more appropriate for services such as postal collection. This added dynamic environment on top of meeting time window constraints is a situation encountered in business such as with e-commerce, where order fulfillment centers around customer demand and expectations (Slater 2002). Slater tackled this problem using the methods of demand forecasting that create phantom orders and routes based on previous fulfilled services. The phantom orders and routes are then replaced with actual orders and subsequential phantom routes will be calculated based on these new orders. His system is centered

The time-dependent Traveling Salesman Problem (TDTSP) is an extension of the TSP that takes into account the phenomenon of urban congestion (Malandraki and Daskin 1992) and is the focus of this thesis research. Previously, the TSP was modeled by assuming constant travel times over the network. This does not reflect the impact of temporal cycles of fluctuating traffic density on vehicle travel speeds (Malandraki and Daskin 1992). Ichoua, Gendreau and Potvin (2002) state that in order to improve the application of these models to real-world problems, the aspect of the time dependency of travel times must be incorporated into the model. Hill and Benton (1992) go further to argue that a model based on constant travel speeds will result in solutions that may not be optimal.

The TDTSP has been studied in literature with various algorithms developed by Fox, Gavish, and Graves (1980); Malandraki and Daskin (1992); Picard and Queyranne (1978), and Hill and Benton (1992). Bowman (1956) considered this generalization of the TSP as early as 1956. The TDTSP states that the travel cost between city i to city j depends on time period given by t. There are n time periods (t) and travel cost becomes a function of not just distance between (i,j) but also of time.

Fox, Gravish and Graves (1980) and Picard and Queyranne (1978) were among the first to formulate the TDTSP. Their work makes the assumption that each node is in its own time period. Thus, the number of time periods is the same as the number of nodes to be visited in the tour. Another way to express the TDTSP is to treat time as divided into a set number of periods (t) of deterministic length. The cost is then a step function based on the departure time at the origin of link (i,j). Assume the TDTSP is on a directed graph $G(V,E)$, where V represents the set of nodes and E

the set of edges or links. A directed link is given by the pair *(i,j)*. An *n* x *n* time dependent travel matrix $C(t) = [C_{ij}(t_i)]$ for each time period, *t*, is given to represent the travel times on every link *(i,j)* that is in *E*. The cost, C_{ij}, is a function of departure time, t_i, from the origin node *i* of the link *(i,j)*. This approach to the TDTSP treats cost as depending on the time of day and not by the sequence of node visitation (Malandraki and Daskin 1996), which is more realistic than assuming constant travel speeds. The theory behind this approach is to create a link per time period, (*t*), for every (*i,j*) pair. Figure 1.2 is an example of the visual representation of this approach for a network of three time periods. Time period 1 in green, period 2 in black and period 3 in blue.

Ichoua, Gendreau and Potvin (2002) classified the variations in methods for handling time-dependent travel speeds into four categories through an extensive literature review of the TDTSP. The first is a "simple" travel time function that requires manual user intervention. The second is based on discrete travel time step functions, where the time period is divided into time intervals and is basically an

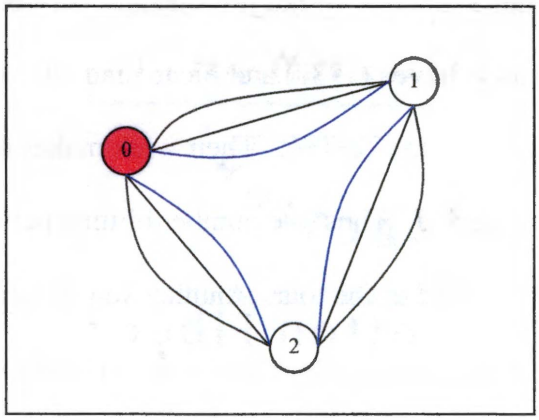


Figure 1.2 Visual representations of three time periods.

requires manual user intervention. The second is based on discrete travel time step functions, where the time period is divided into time intervals and is basically an approximation method. The third uses continuous travel time functions and more closely resembles reality. These continuous functions are restricted in their tractability and usually result in some degree of an approximation. The final category uses Markovian formulations of travel time. These formulations are based on probability functions. Adding the time-dependent constraint to the TSP model drastically improves it over other models based on fixed travel times (Ichoua, Gendreau and Potvin 2002). The TSP model is also improved in its usefulness to businesses and organizations that depend on these logistics (Hill and Benton 1992) by accounting for these temporal cycles.

Another extension to the TSP accounts for stochastic travel events that also affect travel speeds. Variations in travel speeds have been modeled for predictable events, such as rush hour, but can also be affected by unpredictable events, such as accidents, vehicle breakdowns and weather (Ichoua, Gendreau and Potvin, 2002). These stochastic components (Malandraki and Dial, 1996) also contribute significantly to the variation of travel speeds. Despite the possibility of such unpredictable events mentioned as causes of travel speed variations in the literature, they have not been widely incorporated into the TSP model. Park and Song (1997) mention this effect on speed due to traffic congestion and other random events but neglect these random events in their model. Miller, Wu and Hung (1999) developed a GIS-based traffic congestion model for time-critical logistics, such as "just-in-time" (JIT) systems that include a module to perform "what if" scenarios such as time-

independent events, which they term unplanned disturbances. They deemed these "what if" scenarios vitally important due to networks functioning at already near capacity and which are therefore more susceptible to congestion from unpredictable events (Miller, Wu and Hung 1999). Their methodology is to develop hypothetical scenarios and to predict the impact on the JIT schedules as well as to develop robust strategies to ensure the deliverables are still met. Their modeling focused more on developing effective traffic assignment patterns through equilibrium analysis, while the objective for the TSP is to minimize the cost of the route chosen. Travel speeds in TSP models are calculated by either a deterministic function (Malandraki and Dial, 1996) or by a matrix or array structure containing the travel speeds through the different time periods of the day for the links in the network. This could be based on past data or a step function that depends on the time of day.

Miller, Wu and Hung's (1999) GIS-based traffic model is an excellent example of expanding the scope of the GIS problem domain as well as increasing the power of routing algorithms. The GIS handles the complex spatial (the street arcs and junctions) and temporal features (varying travel speed data) of the underlying transportation network that routing algorithms require for finding a solution. The benefits of combining algorithms with graphic interfaces to take into consideration the spatial component of these problems (Keenan, 1998) began to be recognized in the 1980's. Routing software such as ArcLogistics and TransCAD are examples of this movement to continue the development and application of these routing algorithms. These software packages are restricted to handling more general-purpose issues, and manual intervention is necessary for users who want to account for real

world situations that are not part of their generalized mathematical formulation (Keenan 1998). These software packages offer no means to provide more custom and precise tools for handling temporal concerns. Miller and Shaw (2001) state that GIS vendors understand that it is not possible to provide all transportation users with the specific and various tools they require in solving very specific problems, such as the TDTSP. Instead, vendors such as ESRI have taken the path of developing a software environment that allows interoperability and customization by the users. This environment allows the problem domain of ArcInfo 8 to be expanded to include the appropriate tools to handle the TDTSP.

The TDTSP is viewed as a spatial problem (Keenan 1998), and GIS facilitate such a view. The GIS spatial environment provides visual tools for analysis and display of the routing algorithm results and is an excellent platform choice on which to solve the TDTSP. Sutton (1997) focuses on the ability of GIS to provide a uniform environment for DBMS, spatial analysis and visualization. He also states that GIS spatial environments are much better equipped to mange and visually represent transportation networks than other non-spatial transportation data models. Even though the TSP and its extensions have been well documented and researched, thier incorporation into such products as ESRI's ArcInfo 8 have not been realized by the market.

The main purpose of this thesis is to use the architectural framework of a GIS, coupled with a TDTSP module that incorporates stochastic travel events into its solution, to continue to extend the applicability of the TSP model to the real world. ArcInfo 8's open Component Object Model (COM)-based architecture allows me to

take the specific problem domain of the TDTSP and to implement appropriate tools to mange and solve the problem. ArcInfo allows componentware software modules to be developed that provide a seamless integration of the routing algorithm with the GIS environment to the user. In this way I can tailor ArcInfo 8 to solve the TDTSP problem as well as identify how time-independent travel events can affect the tour.

The primary objective of my thesis research is to implement within ESRI's ArcInfo 8 a Traveling Salesman Problem that takes into consideration travel speeds that are a function of time, and are affected by stochastic travel events. There are many types of stochastic events but the focus of this thesis will be traffic accidents that directly affect the street network. The TDTSP with stochastic travel events will be tested on a restricted street network derived from the Washington D.C. metropolitan area. The TDTSP routing algorithm will be based on Malandraki and Dial's (1996) restricted dynamic programming heuristic and will be integrated into ArcInfo as a COM object. My strategy to handle the stochastic events in a GIS environment is the beginning step towards a TSP model that can handle dynamic temporal travel speed variations effectively and thus provide more realistic solutions. My approach focuses on determining if a re-ordering of the sequence of node visitation provides a better alternative to adhering to the original TDTSP tour. This Re-Order algorithm is also based on Malandraki and Dial's (1996) heuristic and will be accessed in ArcInfo 8 through a custom COM object.

Chapter 2

Literature Review

The Traveling Salesman Problem (TSP) is the basis upon which the TDTSP is built. On link 1 he travels from the depot to some other city, and on link *n,* he travels to city 1. The formulation of the TSP has the binary variables δ_{ijk}, which have a value of 1 if the salesman travels from city *i* to city *j* on link *k,* and 0 otherwise. Based on Lawler et al. (1985), the TSP can be formulated:

$$\text{Min} \sum_{i,j,k} C_{ij}\, \delta_{ijk} \qquad (1)$$

subject to $\sum_{i,j} \delta_{ijk} = 1, \quad k = 2, \ldots, n-1$ (2a)

$$\sum_{j} \delta_{1j1} = 1, \qquad (2b)$$

$$\sum_{i} \delta_{i1n} = 1, \qquad (2c)$$

$$\sum_{j,k} \delta_{ijk} = 1, \quad i = 2, \ldots, n-1 \qquad (3)$$

$$\sum_{i,k} \delta_{ijk} = 1, \quad j = 2, \ldots, n-1 \qquad (4)$$

$$\sum_{i} \delta_{ijk} = \sum_{j'} \delta_{jj'k+1} \qquad (5a)$$

$j = 2, \ldots, n\text{-}1; \qquad k = 2, \ldots, n\text{-}1,$

$$\delta_{1j1} = \sum_{j'} \delta_{jj'2} \qquad j = 2, \ldots, n\text{-}1; \qquad (5b)$$

$$\sum_{i} \delta_{ij,n-1} = \delta_{j1n}. \qquad (5c)$$

Constraints (2a, 2b, 2c) ensures that there is only one link for every pair *(i,j)*. It also guarantees that the first link starts at the node 1, the depot, and the last link *n* must end at node 1. Constraint (3) states that only one link can leave the origin node *i*. Constraint (4) goes along with (3) and states that only one link can enter the destination node *j*. The purpose of constraints (5a and 5b) is to ensure that the TSP solution is one tour and if it was broken into two subtours there are two arcs connecting the two subtours making it a complete tour again. The first part states that the sum of links for all *(i,j)* pairs must equal the sum of links leaving *j* and going to *j'*. The second part guarantees that not only is there only one arc leaving node 1 but also there is only one arc leaving the destination node of the link pair *(i,j)* and the last part ensures that one arc returns to node 1.

There are many resources on the Internet for solving the TSP, such as journals, instances of TSP, and even open source code. The TSPLIB website, maintained by Gerhard Reinelt (2001), is a library of sample instances of the TSP. The TSP instances include the symmetric TSP, asymmetric TSP and sequential ordering problem (SOP). The TSPBIB website, created by Pablo Mascato (2000), is an excellent resource for researching the numerous options to solve the TSP, such as Genetic or Tabu Search approaches, as well as various extensions of the TSP. It includes software links, some like Concorde are free to download, journals, Java demos of problems and reviewed data structures. Open source code to solve the TSP can be found at GLPK, which stands for GNU Linear Programming Kit. This kit is for large-scale linear programming, mixed integer programming and other related problems. Another source for code is BonsiaG, which is also for mixed-integer linear

programming problems. Problems to be solved by BonsaiG must be in MPS format, which is an industry standard for describing linear problems.

2.1 Review of Literature for the TDTSP

Computational complexity theory places the TSP in the classification of *NP*-complete problems (Lawler et al. 1985). The TDTSP also belongs to the *NP*-complete problems, and due to this it seems unlikely that a polynomial-time exact algorithm can be developed (Malandraki and Dial 1996) without overwhelming computation time. The difficulty inherent with this problem leaves the options to solving the TDTSP by either a slow solution method that results in optimality, a fast solution method that gives sub-optimal results, or some middle ground between the previous two options. The efficiency of an exact algorithm reaching an optimal solution deteriorates as the number of nodes to visit becomes large (Lawler et al. 1985), especially since it is a $O(2^n)$ problem. Approximation methods offer the best possibility to finding near optimal tour to the TSP (Solomon, 1987) but exact algorithms have been formulated to solve restricted aspects of the TDTSP.

Picard and Queyranne, (1978) and Fox, Gavish and Graves (1980), were the earliest to formulate exact algorithms for the TDTSP. Picard and Queyranne's research focused on applying the TDTSP to the tardiness problem in One-Machine Sequencing. Their three integer programming formulation falls into the discrete time step function classification as mentioned by Ichoua, Gendreau and Potvin (2002) reviewed in Chapter 1. Picard and Queyranne argue that the cost associated with

setting up the order of jobs on a machine is not dependent on the second job *j* of the pair *(i,j),* but on the preceding job *i* and the position in time that *(i,j)* hold in the sequence. Therefore, the problem can be reduced to an assignment problem. The discrete time intervals are centered under the assumption that each pair *(i,j)* is in one time period (see Figure 2.1). This assumption is acceptable in One-Machine Sequencing problems but when applied to TDTSP tours on real-world transportation, the assumption that each pair *(ij)* is one time period may not hold true. Picard and Queyranne formed relaxations on the problem to establish lower bounds that are used in an exact branch and bound procedure to solve the TDTSP for 15-20 jobs. Picard and Queyranne are restricted to a small size problem domain to solve due to the overwhelming complexity in achieving an exact solution.

Fox, Gavish and Graves (1980) also make the assumption that travel time between any two cities (nodes) is during one time interval. They also fall into the

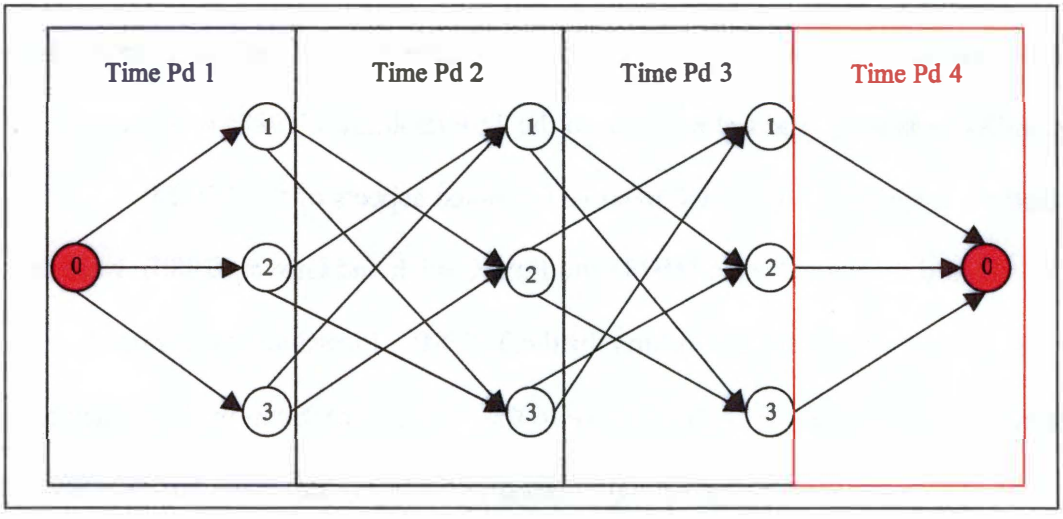


Figure 2.1 A TSP tour that assumes any link *(i,j)* is during one time period.

category of handling time-dependent travel time as a function of discrete time intervals. Figure 2.1 is the visual representation of their TDTSP, which assumes one time period per node sequence. Their research improved the formulations of the linear constraints for the TDTSP. Picard and Queyranne (1978) developed a formulation that only needed $2n$ constraints, but it required a quadratic objective function, which is not as easily solved as Fox, Gavish and Graves' linear version. Picard and Queyranne's original mathematical formulation of the TDTSP is stated below. There are n nodes and n connecting arcs. The travel cost, C_{ijk}, is determined by the distance between city i to city j and what poisition this link, k, is in the tour. This formulation assumes that the tour orginates at the depot in time period 0 and returns tot he depot in time period n.

The objective is to:

$$\text{Min} \sum_{i=1}^{n} \sum_{j=1}^{n} \sum_{k=1}^{n} C_{ijk}\, \delta_{ijk}. \qquad (6)$$

$$\text{subject to} \sum_{j=1}^{n} \sum_{k=1}^{n} \delta_{ijk} = 1, \quad i = 1, \ldots, n, \qquad (7a)$$

$$\sum_{i=1}^{n} \sum_{k=1}^{n} \delta_{ijk} = 1, \quad j = 1, \ldots, n, \qquad (7b)$$

$$\sum_{i=1}^{n} \sum_{j=1}^{n} \delta_{ijk} = 1, \quad k = 1, \ldots, n, \qquad (7c)$$

$$\sum_{j=1}^{n} \sum_{k=2}^{n} k\delta_{ijk} - \sum_{j=1}^{n} \sum_{k=1}^{n} k\delta_{jik} = 1, \quad i = 2, \ldots, n, \qquad (8)$$

$$\delta_{ijk} = 0 \text{ or } 1, \quad \forall\, i,j,k$$

Constraint (7a) states that exactly one arc enters each node and (7b) states that only one arc leaves each node. Constraint (7c) goes further to ensure that for each pair *(i,j)*, there is only one associated link k. Fox, Gavish and Graves (1980) showed that the n^3 variables and $4n$-1 constraints in the above formulation could be reduced by replacing (7a, b, c) with:

$$\sum_{i=1}^{n} \sum_{j=1}^{n} \sum_{k=1}^{n} \delta ijk = n. \qquad (7d)$$

Constraint (8) states that all subtours must include node 1 and therefore only one subtour can be formed which is a complete tour solution.

Lucena (1990) focused on generalizing the TDTSP problem for the Deliveryman Case. The method Lucena uses is based on Picard and Queyranne's (1978) relaxation techniques to acquire lower bounds for using in the branch and bound procedure called the tree-search. The intent is to be able to apply this exact algorithm to a larger problem size. This is a Lagrangian relaxation technique for finding bounds that are further split into a number of components. A Lagrangian relaxation is taking some constraint and removing it from the model, such as relaxing the constraint that only one link can enter node *i,* and then adjusting the objective function (Ahuja, Magnanti, and Orlin 1993). This is a common technique for procedures that uses bounds on problems. The Lagrangian method uses multipliers to tighten the relaxation and achieve better bounds. Lucena (1990) uses a subgradient optimization method to control the direction of the relaxation multipliers. Instead of the overall Lagrangian bound being optimized, it is these individual components of the whole that are optimized. This is typically classified as a local optimization

method and the resulting tour will be a local optimal solution and not a global optimal solution. Time-dependent travel costs used for this formulation of the TDTSP are dependent on where the link falls in the sequence of the tour, which is very similar to Picard and Queyranne's (1978) approach. This procedure was successfully solved for networks up to 30 nodes with the travel distance (time) cost between nodes being based on a Euclidean plane instead of a real transportation network. A Euclidean plane is not a good indicator of this algorithm's performance on real-world network which is much more complex. The problem size is still relatively small and if one wants to solve for a larger problem size based on a real world network, one must turn to heuristic approaches.

Malandraki and Daskin (1992) formulated a mixed integer linear programming formulation of TDTSP with travel time based on a step function. This function considers the travel cost between two customers (nodes) to be dependent on distance and time of day. The presence for time windows is also considered in the formulation. Several heuristics are developed and analyzed for the TDTSP and the time-dependent vehicle routing problem (TDVRP). The TDTSP is equivalent to a TDVRP problem where the number of vehicles is equal to one. The heuristics developed are variations of the nearest-neighbor construction procedure for the TDTSP and the TDVRP and a cutting-planes heuristic for the TDTSP, which can only solve very small problems. The nearest-neighbor variation that proved to be the best overall was based on applying the algorithm n-1 times to each node to be visited next and then the best solution found is chosen. The probabilistic heuristics of the nearest-neighbor procedure are theorized to find a better solution if applied for a

longer time. Typically the nearest-neighbor heuristic is associated with an improvement heuristic such as the k-opt exchange procedure to improve the solution. Malandraki and Daskin (1992) argue that when applying the k-opt exchange procedure to the TDTSP, it becomes computationally expensive. This is because the travel times of more than one link may change due to changes in traveling direction or because the starting time may change, especially when considering time windows. Therefore, no improvement procedure was applied to initial solution found by the construction procedures.

Bianco, Mingozzi and Ricciardelli (1993) consider a very special case of the TDTSP where the objective is to minimize the cumulative cost of all distances traveled from the origin to all the other nodes with travel costs also treated as a step function. Two exact algorithms based on Lagrangian relaxations are presented that are very similar to Lucena's (1990) exact tree-search procedures. Also presented is a heuristic procedure based on dynamic programming methods. Dynamic programming is a stagewise optimization technique (Ahju, Magnanti and Orlin 1993) based on the possible combinatorial enumerations of a problem. Dynamic programming procedures are computationally explosive in the number of enumerations they store. To reduce the "state space", as Bianco, Mingozzi and Ricciardelli (1993) call it, in the dynamic programming algorithm, a heuristic procedure is developed to evaluate the distance from optimality of the solution so as to form a bound. The algorithms were used on Euclidean planes. Optimality was reached for the two exact algorithms for only when n equaled less than 35 and for the

heuristic up to 25 nodes. The heuristic was shown to achieve a near-optimal solution (within 3%) for n up to 45.

Hill and Benton (1992) proposed a model based on time-dependent travel speeds for the TDVRP. Since, the TDVRP turns into the TDTSP when the number of vehicles equals one, this is also very applicable for the TDTSP. They formulated travel time on a given link (i, j), starting at time period T, as:

$$dij / rijT = (riT + rjT) / 2 \qquad (9)$$

Where

dij = distance between i and j.

$rijT$ = average speed from i to j during starting period T.

riT = average speed associated with "area" around location i at period T.

This formulation is based on indexing by time periods and nodes. This time-dependent formulation is also the first to model variations in travel speeds that can occur along the link. These travel speeds, $rijT$, can be estimated in three ways according to Hill and Benton. The first way is through actual drivers making subjective estimates of the average travel speed in the areas by time of day. The second way is through historical data available. The last way to determine travel speeds is through actual travel time data. Hill and Benton's (1992) model was applied to real world situations in a number of metropolitan areas and showed the validity of the use of time-dependent travel speeds.

Malandraki and Dial (1996) formulated travel cost as a deterministic function of departure time t_i, from the origin node i of the link in their TDTSP heuristic

algorithm. Their procedure also falls into the classification of discrete step function in how it handles the dynamic cost component. They modify a dynamic programming exact algorithm to avoid its characteristics of exponential time and space requirements. Malandraki and Dial use a restricted dynamic algorithm that, at each iteration, only saves a user specified number of partial tours. If the user specifies to save only one partial tour for each stage of the tour construction, then this becomes the classic greedy nearest-neighbor heuristic algorithm. They tested the restricted algorithm for user-specified size of 1, 100, 1000, 5000 and 15000 partial tours with nodes ranging from 10 up to 55. As expected the algorithm solution improved, in respect to reaching the optimal solution, with the increase in the number of saved partial tours. This heuristic also outperformed previously known heuristics in solution quality, execution time, and size of the problem solved.

Ichoua, Gendreau and Potvin (2002) state that continuous travel times are more accurate in modeling real world phenomena but are not pragmatic for model formulations and implementations. Therefore, Ichoua, Gendreau and Potvin focus on the issue of enforcing "first-in-first-out" (FIFO) property for the time-dependent models. This is achieved by indexing travel speeds by time periods and arcs instead of the traditional view of indexing by time periods and nodes as formulated by Hill and Benton (1992). Travel speeds along an arc can now be seen not as constant but as being affected by the crossing of time period boundaries resulting in variations in speed along an arc. Ichoua, Gendreau and Potvin's time-dependent modeling approach is implemented through Taillard et al.'s (1997) parallel tabu search heuristic. Tabu search starts with an initial solution and tries to improve it with an

iterative local neighborhood search process. Tabu search is a complex procedure and to reduce calculation, an approximation is used on the local neighborhood search. Then M best solutions are determined and are then evaluated exactly to determine the best solution. The value of M was set from 4 up to 40. As M increased, so did the run time while the objective value decreased. Ichoua, Gendreau and Potvin claim that these increases in run time are small. On the whole, the results showed that time-dependent travel speeds proved to find a more realistic solution than that of static travel speeds.

The literature review of the TDTSP problem verifies that it is a significant and mandatory extension to solve the TSP under realistic conditions. Despite the added computational complexity in solving the TSP with time dependent travel speeds, it has been shown that there are exact algorithm solutions for small problems, and for more complex problems there are several effective heuristics. It is interesting to note that in the articles by Ichoua, Gendreau and Potvin (2002), Malandraki and Dial (1996), Park and Song (1997) and Miller, Wu and Hung (1999), they all state that the variations in travel speed is not only due to predictable events, such as congestion, but also from stochastic events, such as traffic accidents. Despite this acknowledgement of travel speed variants being due to two different types of events, only Miller, Wu, and Hung try to address both. Their approach to the stochastic travel events is to depend on users to develop potential situations that may disrupt the JIT schedule through querying the results. A better approach would be to process these disruptions as they occur and to determine if an alternative solution is available to avoid delays.

This addition of considering the effects of traffic accidents on the cost performance of the tour is the focus of my contribution to the TDTSP model.

2.2 Review of Literature for GIS and Routing

Lewis (1990) states that historically transportation analysts have been among the first users of computerized-based tools. This is primarily because since the 1950's, and in response to federal policy, the field has developed approaches to solving such problems as land-use and transportation systems (LUTS), which rely heavily on computer-based methodology. Lewis mentions that GIS systems have been around in some form since the 1950's, though they are principally a product of the 1980's. Therefore, it is a natural adaptation of transportation problems to create the field of GIS-T. He also states that the growing use of GIS is due to perceived notions that GIS provides a means to increase productivity, effectiveness, and quality as well as reduce cost. Sutton (1997) supports Lewis' claims of the potential to benefit transportation planners and transportation operational researchers. The underlying implication is that these achievements are a result of the visualization techniques that are the core of GIS software.

The validity of such claims can be found in the discipline of Image Theory (IT), which is area of graphics research. According to Crossland et al. (1995), IT builds on efficiency as the basic premise of determining the effectiveness of different visual structuring aids. Crossland et al. conducted a literature review in Decision Support Systems (DSS) and Information Science that found applying a GIS in

problem formulation and solution finding to be extremely beneficial. Their research showed that a Spatial Decision Support System (SDSS)/GIS is an enabler for providing better decision-making. Using IT insights as visual structuring aids, their study showed that users of a SDSS/GIS solved problems quicker and more accurately. Such systems can be useful for most business decisions since the majority of the data required for a decision is made up of more than one spatial component, such as customer locations.

Bodin and Levy (1994) implemented a visual structuring aid in their design of computer-assisted Vehicle Routing Schedule (VRS) problem in a GIS environment. The design is aimed at the user finding a solution with the VRS/GIS and possibly tailoring the solution as needed. To be able to do this effectively, Bodin and Levy state that the requirements are high quality graphics, excellent algorithms, detailed reports of the solution generated, manual intervention capabilities, and of course a GIS. This visualization of the VRS through GIS allows the users to manage complex algorithms in solving their problems as well as being able to implement improvement. Bodin and Levy assume that knowledgeable expert users are the norm, though this may not be a realistic assumption.

By the 1980's, there was an increasing recognition of the benefits of combining algorithms with graphic interfaces to take into consideration the spatial component of these problems (Keenan, 1998). Routing software such as ESRI's ArcLogistics and TransCAD are examples of this strategy though they are limited to handling more general-purpose issues and are aimed at expecting user intervention to adjust the solution returned by these packages to account for real world events, such

as dynamic ordering and congestion. ArcLogistics handles these situations through manual intervention. At the beginning of the work period the known orders are loaded into the software and the route paths are created. If new orders come in, they are added and the routes are adjusted to accommodate the new orders. If the time window constraint cannot be made to fit in with the existing route, the order is saved till the next work period. The ArcLogisitcs user interface also allows for manual manipulation of the routes and orders. These packages demonstrate how GIS functions as a means to display and manipulate spatial information as well as a database management system (DBMS) to organize and query data.

Routing algorithms focus on generating an optimal solution. Jones (1994) points out that optimization issues are not just about developing a fast and clever algorithm, but rather about developing appropriate representation of models, algorithms, data and the solution. This is so the user can design, validate, debug and understand the models and algorithms. Jones argues that the algorithms are only part of any problem. He views the process of problem solving as a series of stages and each stage may require its own visual representation of the problem. These stages are identification of problem, formulation of the model, data collection, development of algorithm, solution validation and presentation of the solution. A GIS system can assist with these stages. Jones looks into the scientific field of Visualization for developing interactive computer graphics. He uses the example of the originator of the spreadsheet, Dan Brickin, to show how visualization provides insightful representations for difficult problems. The spreadsheet representation style Brickin developed was due to focusing on how people actually visually represent their

problems. Jones claims that the main benfit is this combination of algorithms with a simple representation of the data in tables. Jones goes further to say that visualization is not just for output but also for input, model formulation and data validation. Another benefit of graphic representation is non-technical users are now able to implement algorithms and models, whose underlying concepts they understand, without knoweledge of the actual mathematical notation. Routing problems are a spatially viewed problem and coupling them with GIS can provide the benefits to users mentioned by Jones.

Barbosa and Hirko (1980) agree with Jones (1994) that there are well known difficulties for non-analyst/non-technical individuals using algorithms and/or models. Therefore, Barbosa and Hirko focused on the graphic interface between the user and the algorithm used by their DSS. They argue that algorithms can be even more effective when they adopt some of the inherent qualities of a DSS. These qualities are interface, flexibility, control and feedback. The flexibility, control and feedback qualities are crucial parts of the paradigm of a DSS. They used these qualities to develop a DSS with an integrated algorithm. The DSS, according to Miller and Shaw (2001), generates and compares alternative solutions. Decision-makers then make the final selection based on the details of each alternative. The user interface dimension presents a common environment while the DSS links to other technologies to solve the problem in the background. This is achieved through different software coupling strategies. Although Barbosa and Hirko support the idea of difficulties non-analysts have with understanding the formulation and procedures of the algorithms, their design of the DSS is based on the user having the ability to have full control of the

algorithm. The design of the interface should be constructed so that the algorithm is made understandable to non-analyst. Barbosa and Hirko give no indication of the success of this aspect.

Tarantilis and Kiranoudis (2002) also took the approach of having the user interface be the only part of the SDSS visible to the user. They used a GIS as the background information system for the SDSS and interact with peripheral software tools to come up with solutions to the Vehicle Routing Problem (VRP). The GIS environment provides excellent visual tools for analysis and display of the routing algorithm results by the decision-maker. Tarantilis and Kiranoudis defend the use of an SDSS for VRP problems by citing that as early as the 1980's, the VRP models were seen as DSSs. At the same time, GIS was becoming popular in the market place. The user can benefit from viewing information in a different way when the GIS are used as the framework for an SDSS. The decision-makers can view each route for each vehicle with their associated attributes of cost and capacity. The decision-maker can then view if any modifications are required based on their experienced expert knowledge, a required assumption of an SDSS, of the service region. Tarantilis and Kiranoudis list some of the benefits of using commercial VRP SDSS, such as reduction of transport cost, support of semi-structured or unstructured decisions, tighter control of distribution as well as less need of high skill level of individuals. I assume that the high skill level Tarantilis and Kiranoudis refer to concern knowledge of mathematical notation of the routing algorithms. However, I would interpret high skill level to reflect the knowledge base of the user for the entire problem, which is an essential skill for a user of this SDSS.

Keenan (1998) showed it is necessary to create a synthesis between GIS and routing algorithms to create a SDSS. He states that the TDTSP is viewed as a spatial problem by the decision-maker, and the GIS facilitates such a view. A SDSS also requires the ability to store, manage and display spatially referenced data that is needed to solve the routing problem. In the 1980's data storage and management was handled by DBMS, but all of these functions are available in a GIS, in addition to the power of visualization and spatial queries. The Crossland et al. (1995) study also concurs that coupling a GIS with an SDSS allows visual problem structuring to result in users solving complex problems in shorter time and with fewer errors than those without a GIS. Keenan defends his SDSS/GIS through his analysis of previous DSS applications, which confirm that routing was the most important area for businesses usage of DSSs. He claims that vehicle routing is less suitable for expert systems, which are a sub-area of Artificial Intelligence, approach because of the importance of user's skill and knowledge of the problem domain.

A SDSS that uses a GIS as the framework also needs a component to do the analytical routines to generate alternative solutions. Coupling GIS with modeling or statistical software is necessary due to each type of software having its own set of valuable functions not found in GIS (Nyerges 1992). Through a coupling strategy, GIS can gain access to analytic tools outside its problem domain and the modeling software is able to visually represent itself through GIS graphic capabilities. Nyerges says that through this 'borrowing' of functions the GIS becomes a more effective problem-solving environment for the user. The coupling strategy is also cost efficient

because it minimizes the redesigning and development of the tools the GIS or the other software are lacking to achieve the same effect.

Goodchild et al. (1992) state that GIS have great potential to support many different types of analysis modeling. The reasons are that space is another way to index data, spatial displays are perfect mediums for gaining information on relative location of objects, GIS overlay capabilities can link seemingly unrelated events, and distribution of spatial objects is an important factor in determining object interactions. They also point out that any spatial analysis software requires a means to input, store, manage and display spatially referenced data, which are available in a GIS. To achieve the integration of GIS and algorithms and/or models, Goodchild et al. identified four coupling strategies. The first is the standalone approach. It requires finding software that has been specifically designed to solve routing problems or creating your own software package for solving these problems. This could be because either the analysis required by the algorithm is completely different than the standard GIS, or the algorithm requires a specially tailored data structure that is not available in a GIS due to the GIS's need to be flexible for handling many forms of problems. Some examples of this coupling strategy are FIELDS (Ralston and Ray, 1987), TRAILMAN (Ralston, Liu, and Zhang, 1993) and the Bangladesh Transportation Modeling System (Ralston, Tharakan, and Liu, 1994). The second approach is called loosely-coupled, and involves using separate software packages that are tailored for a specific area of the problem at hand. An input-output process mitigates between the GIS and the routing algorithm. The disadvantage to this approach is that the only information that can be accessed by the separate processes is

what is stored in the interchange files. The third approach is tight-coupled integration that can modify operations or create custom operations within the GIS software. Goodchild et al. state this can be done through macro languages offered by GIS packages allowing access to the standard user interface and data. The last option is full integration. This involves the GIS software company creating their own algorithm solver that they would be available to every user as part of the GIS. They mention that the difficulties lie in the fact that the inclusion of such algorithms might call for a major reconstruction of the GIS internal framework, which would only be advantageous to those who use these new algorithms.

Loose coupling is the most popular approach and the easiest for the user to implement since it is just a matter of reading and writing input and output files. An example of this strategy is Wu, Miller and Hung's (2001) GIS-based decision support tool for modeling dynamic congestion and providing the ability to route through the network. The dynamic congestion module is a stand-alone system that reads and writes ArcInfo INFO files. This allows the GIS software to manage the data and the visualization. They state that there is a need for these tools due to the complexity of traffic patterns, which can cause businesses to suffer loses, especially in "just-in-time" (JIT) systems. With the increasing power of computational platforms, the GIS data model can be taken advantage of to model the dynamic flow patterns that better represent the real world through coupling strategies.

Bennett (1997) states that loose-coupling integration strategies fail to provide a consistent user interface and data structure, as would a fully integrated system. Loose-coupling also does not provide support for development and modification to

the models for more user-defined purposes or allow for user interaction, especially in simulated events. Therefore, Bennett argues for tight-coupling systems to allow the GIS user the capability to interact and control the model through software 'hooks' and/or macro-languages provide by the GIS environment. This strategy is used by Bennett to develop a geoprocessing environment consisting of a GIS, computer simulation methods, and model-based management to form a SDSS. This allows the user to visualize ongoing simulations and be able to halt the simulation to gain more details of the underlying processes and investigate relationships between objects. The simulation purpose is to model dynamic spatial processes through space and time. Bennett says there are three components to this model: state, process and relation. The state(s) is represented in the GIS as geographical fields or objects. The processes describe how states change through time. The relation part of the data model tries to capture how the geographical system interacts across space. The objective of modeling state, process and relation is to follow geographical entities that come into existence and then go out, and which are interacting with the temporal landscape. Relating that to the TDTSP, these entities would be such time-independent events as traffic accidents that effect travel speeds until these accidents have been removed from the scene.

Ungerer and Goodchild (2002) discuss another coupling strategy available that is based on COM technology. COM standards, created by Microsoft, are designed to enhance software interoperability. This means that, for example, that GIS software can access COM components of other software, such as Excel, to perform statistical routines available in that software. COM standards go a step further and

allow components written in different languages to communicate directly with each other. The only constraint is that the programming language must be a COM-complaint language such as VsC++, VB or Delphi. COM-complaint software allows direct access to macro programming, with Visual Basic for Applications (VBA), from within its own environment and makes tight-coupling for users much easier to develop and implement. VBA is a simple programming language that provides the means to develop and deploy custom tools quickly. Using this COM-complaint software, Ungerer and Goodchild are able to create custom-tailored applications.

Exploring ArcObjects (Zeiler 2001) is an excellent reference to gain a better appreciation of the customization potential available from ESRI's open COM-based architecture. Their specific suite of components objects is called ArcObjects. This component software environment provides the protocol standard to communicate with other component-based software. COM is a methodology of software development and not an object-oriented language. ArcObjects not only allow the developer to build extensions to existing objects but also to access these components in building separate custom applications. In *Exploring ArcObjects*, the history of creating useable and interoperable code is reviewed with COM being the end result of this journey. First attempts at reusable code were created in C++ class libraries and suffered from several limitations when they were distributed. COM objects are essentially a binary unit of code and are therefore more robust in being distributed and in fostering communication between objects or modules. COM is an interface-based programming strategy that defines interaction and communication methods for

software modules. Essentially, interfaces define what functionality an object has and the interface's class defines the procedures and steps to achieve this functionality.

Exploring ArcObjects defines the three ways to write and use ArcObjects. The first method is to use the VBA editor in ArcInfo 8. The VBA editor allows the system to be customized by writing macros. This strategy provides direct access to the essential global variables to manipulate ArcInfo's, *Application* and *ThisDocument* objects. It is also very easy to create custom tools and forms and to quickly integrate them into the GIS environment. The downside is that your code is not protected and is very similar to scripts. A second method is to create an ActiveX COM component such as a DLL or an OCX. Since the reusable code is at the binary level, there is no need to provide source code, header files or even object libraries when distributing your ActiveX control. The final alternative is to develop standalone EXE applications. The disadvantage to these last two alternatives is that there is no direct access to the *Application* and *ThisDocument* objects. These variables are required to gain a 'hook' into the ArcInfo application and the developer must provide the steps to achieve a 'hook' into ArcInfo 8.

With these developing tools available from ArcInfo 8 as well as its framework of a spatial DBMS and a visualization toolbox for display and analysis, I will extend the GIS problem domain to include the TDTSP with stochastic travel events. The implementation of the routing algorithm will appear seamless to the user by utilizing the open COM-based architecture. The user gains greatly from the visual displays of the different stages of the TDTSP from identification of the original problem, the development of the shortest path links to be used to solve the TDTSP, and finally the

display of the routing results. This is in accordance with Jones' (1994) statement that algorithms are only part of the problem solving and an SDSS is actually comprised of a series of stages and each stage may require its own visual representation of the problem. A GIS can assist in facilitating these visualizations. I also will extend ArcInfo's visualization tools and analysis tools to allow the user to query for potential disturbances (stochastic events) to the solution from traffic accidents that can possible affect the cost of the TDTSP tour.

Chapter 3

Data Description and Methodology

As stated in Chapter 1, the purpose of this thesis is to implement a TDTSP problem in a GIS environment. Additionally, the problem will consider both time-dependent traffic events and stochastic events, specifically traffic accidents. By including both types of events, it will be possible to explore how they affect the overall performance and cost of the tour solution.

Stochastic travel events have been considered in the transportation research sub-field of intelligent transportation systems (ITS). Wu, Miller and Hung (1999) mention ITS systems as being able to realistically capture urban traffic flows to provide data for better dynamic models. ITS's are typically aimed at individual users, but this information is also pertinent to commercial users of the transportation network for solving the TSP problem. If an event occurs, affecting one or more street network arcs that are used in a TSP tour, the overall solution could increase in cost. These situations can evolve from such nonrecurring variations in the transportation network (Peetra and Mahmassani, 1995).

In this study I focus on the specific type of stochastic events of vehicle accidents. To prevent the TSP tour from adhering to a much more expensive tour than originally calculated, the nodes not yet reached on the tour can be re-ordered. Such an approach is based on the open-loop adaptive routing rule (OAR). The OAR method is used for handling real-time information to compute routes for in-vehicle

route guidance systems (Fu, 2001). The first step calls for a complete path, or in my case a TDTSP tour, to be computed. According to the OAR, only the first nodes reached prior to ITS reports of updated real-time traffic information are committed to, and the rest of the tour can be re-ordered to provide an alternative sequence to keep overall travel cost minimal.

Hickman and Bernstein's (1997) research is an attempt to account for these random travel events, which they also term as "stochastic". They created a dynamic model for transit service and path choice problems where travel time can be affected by stochastic and time-dependent events. They made the assumption that passengers may receive real-time information on the locations of transit vehicles so that they may adjust their routes accordingly, which is very similar to the OAR approach. Hickman and Bernstein formulated two mathematical versions of the dynamic path choice model. The first is a less-constrained version, which is difficult to solve, and the other is a constrained path choice that is more easily solved. Due to the difficulties of the less-constrained version, the constrained path choice model is used in comparisons to the traditional static version. This model can be related to TDTSP problems due to the way Hickman and Bernstein views the passenger's route as not just a single path but as a route made up of a sequence of segments that can change. The results showed significant differences in path choice and travel time between the static and dynamic models. This verifies that situations may arise from traffic accidents that require a re-ordering of the sequence of nodes to provide a better solution. When compared to the original tour, with the delay times added in from the traffic accident affected tour links, the new re-ordering is a cheaper tour.

Peetra and Matimassani (1995) point out that acknowledging the need to incorporate real-time traffic variation will motivate the development of models that seek to provide more optimal solutions. Wu, Miller and Hung's (2001) *Route Planner* module is an example of supplying optimal solutions to time-critical logistics problems that are highly dependent on accurate timing. This problem framework requires temporal data such as traffic speed variations, which can be managed, analyzed and displayed by software already available in a GIS. Crossland et al. (1995) suggests that such a strategy can possibly lead to solutions of extremely complex problems that were previously unsolvable. Thus, one can come to the conclusion that the TSP can be extended to model intricate real world scenarios.

The TSP extension of stochastic travel speeds has been used to model predictable events, such as rush hour, however travel speeds can also be affected by "unpredictable events", such as accidents, vehicle breakdown and weather (Ichoua, Gendreau and Potvin 2002). Other researchers such as Malandraki and Dial (1996) use the term "stochastic components", which contribute to the variation of travel speeds. Park and Song (1997) used the term "random events" to describe these travel events. Despite the possibility of such unpredictable events mentioned as causes of travel speed variations in the literature, they have not been incorporated into the TSP model.

Miller, Wu and Hung (1999) included a module in their work for "what if" scenarios. This is due to their observation that networks are functioning at already near capacity levels, and therefore are more susceptible to congestion from unpredictable events. As discussed in Chapter 1, their design of a GIS-based traffic

model for "just-in-time" logistics is an example of embedding of routing algorithms within a software designed with tools for collecting, storing, retrieving, analyzing and displaying spatial and geographic information (Crossland et al. 1995). The TDTSP problem has components that can be managed by a GIS. The spatial data that comprise the transportation network are one such component, and another is the spatially referenced data, which make up the customer's locations (Tarantills and Kiranoundis 2002). Routing in a GIS supports a true distance approach instead of a straight-line distance approach based on coordinates in a plane (Keenan 1998). Since TSP problems are commonly solved on directly connected graphs, a "pseudo-network" will be built from shortest path routines on the original network (Miller and Shaw 2001). The cost of each shortest path routine is stored in the travel speed matrices. Each of the shortest paths in the "pseudo-network" may actually be comprised of more than one arc form the original street network. In Figure 3.1, the yellow line represents the shortest path between the two nodes on the street network, and is composed of several links. This yellow line will become one link in the "pseudo-network" and is made of link IDs 1, 2, 8, 7, 3 and 10. These "pseudo-network" links are very similar to ArcInfo 8's concept of the complex edge network feature. Software packages such as TransCAD offer this shortest path network routine. However, ArcInfo 8 does not and it is necessary for me to build custom tools to perform this task. The shortest path costs will be placed in the appropriate time period matrix, and these matrices will become the travel speeds used by the TDTSP to generate a solution tour. GIS also supplies database management system (DBMS)

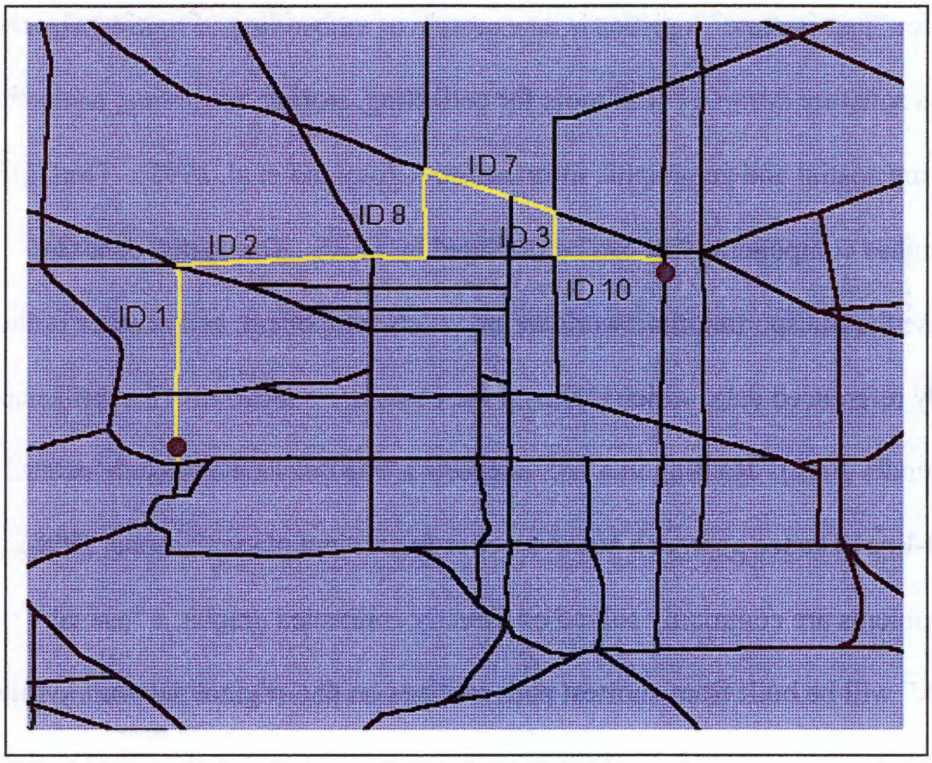


Figure 3.1 Example of one "pseudo-network" link.

capabilities and analysis tools along with the graphic and visual interfaces for creating and managing the implementation of the TDTSP.

3.1 Study Area

The TDTSP with stochastic travel events will be tested on a street network derived from the Washington D.C. metropolitan area. The entire urban street feature dataset for the U.S. was downloaded from the Bureau of Transportation Statistics (BTS) website. The D.C. street network was selected out by using GIS's *Select By Location* function tool to select all street networks that intersected the polygon feature of Washington D.C. This selection reduced the number of arcs to 331 and they are stored in the network feature class *DCstreets*. The D.C. street network is made up of

269 four-lane streets that I will assume for this research to have a speed limit of 55 miles/hour. This dataset also contains 17 one-lane, one-way, streets and 45 two-lane streets. I am simplifying the TDTSP to be a symmetric network and therefore the one-lane streets will be treated as two-lane streets when performing shortest path routines. Further, I will assume the speed limit on two-lane streets to be 30 miles/hour. Since, this study is more concerned with successfully implementing a TDTSP with stochastic travel events in a GIS environment and these assumptions are acceptable. Figure 3.2 shows a map of the study area.

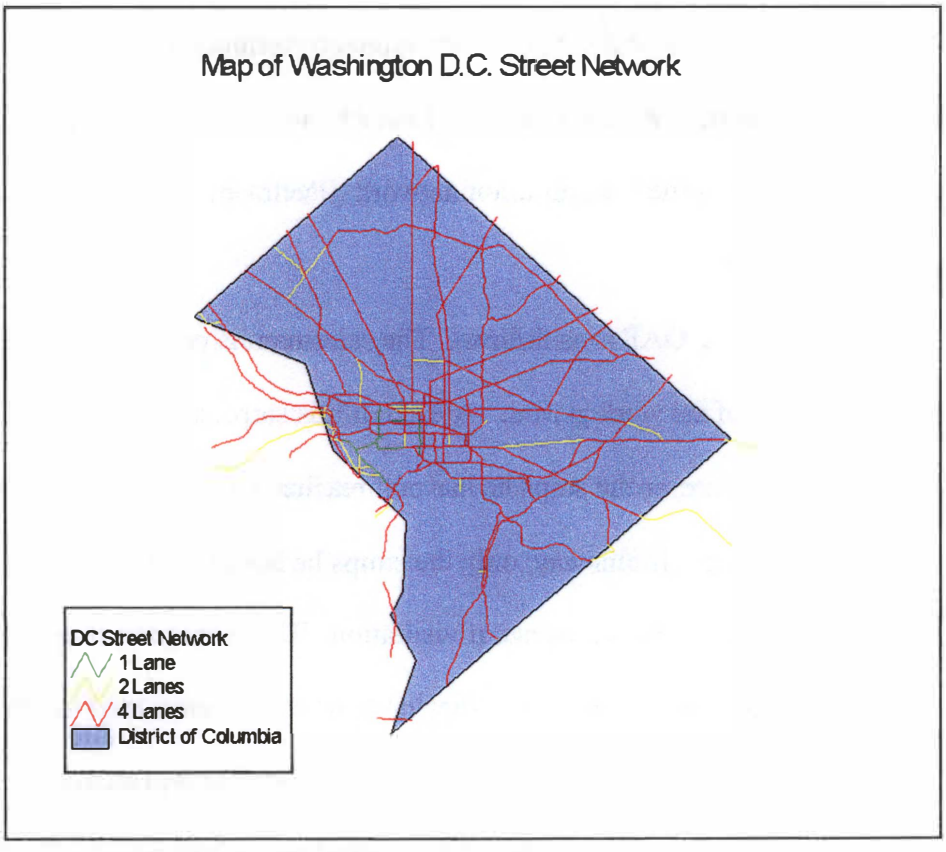


Figure 3.2 Map of study area.

3.2 General Approach

The TDTSP with stochastic travel events will be based on the open-loop adaptive routing rule (OAR). The OAR is a method used for handling real-time information to compute routes for in-vehicle route guidance systems (Fu, 2001). The first step in this method calls for a complete path, or in this case a tour, to be computed. It then assumes that real-time information about travel time events is available as the routed vehicle advances along the tour, and this can cause situations to arise that require a change in the route. Thus, only the first nodes reached prior to these reports of updated real-time traffic information are committed to, and the rest of the tour has the chance of being re-ordered. These situations can evolve form nonrecurring variations in the transportation network (Peetra and Mahmassani, 1995), such as vehicle accidents.

An example of the OAR is as follows. The salesman is provided with the TSP route at the beginning of the work period. At each of his stops, he checks to see if the remainder of the TSP route, or the stops he has not reached yet, has been altered due to stochastic traffic events. In this way, only the stops he has reached at that moment can no longer be altered in their sequence of visitation. The remaining stops retained have the chance of being re-ordered, since they have not been committed to. In Figure 3.3, the red node represents the depot and the yellow line represents the TDTSP tour route. The stops the salesman has reached are in blue and the reaming nodes not yet visited are in green. If an accident occurs on the links not reached and

the link is not one the traveling salesman is using at that moment of this particular traffic report, the sequence of the green stops can still be re-ordered.

In ArcInfo 8 the links selected for the tour include the entire length of these arc segments that make up the links. This can give the impression that the tour paths "double back". In reality the entire arc segment may not be used as demonstrated in Figure 3.3 (circled in gray). This is a minor visuallization problem that does not affect the tour cost calculations.

To achieve the integration of a GIS and a TDTSP routing algorithm, tight-coupling and hybrid-coupling strategies are used. The hybrid coupling is a combination of techniques from loose-coupling and tight-coupling strategies. The

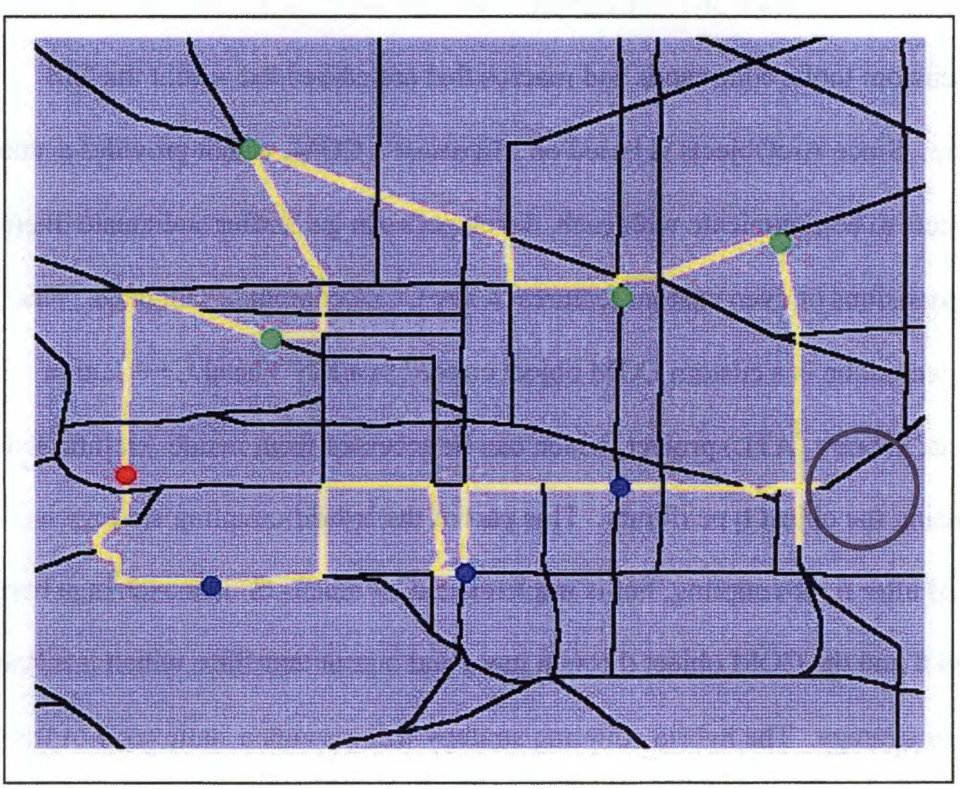


Figure 3.3 A TDTSP tour in progress.

loose-coupled approach involves using separate software that is tailored for a specific aspect of the problem at hand. Then an input-output process facilitates communication between the GIS and the routing algorithm. The disadvantages of this approach are that the only information that can be accessed by these separate processes is what is stored in output files, and also reading and writing to files reduces execution speed. Tight-coupled integration is a seamless transfer of information from the point-of-view of the users. This can be done through macro languages offered by GIS packages, which allow access to the standard user interface and data (Goodchild et al. 1992). Visual Basic for Applications (VBA) is the macro language available that is included in ArcInfo 8. From the VBA editor interface, developers can access ESRI's ArcObjects, a large library of COM objects, in order to create custom tools, commands and macros that are embedded within the GIS software. Since ArcObjects is based on Microsoft's COM, which provides a standard for objects to communicate with each, developers can go further and create their own class extensions or even custom features in any COM complaint language. The hybrid technique is a custom COM object created in a MS Visual C++ Active Template Library (ATL) project, which can be accessed from inside ArcInfo 8 after referencing the object type library. This part of the hybrid-coupling strategy is modeled after tight-coupling, but is implemented by means of data exchange between ArcInfo 8 and the COM object through input and output text files, which is a loose-coupling strategy. The hybrid coupling strategy will provide a static copy of the original travel speeds. When an event occurs that affects travel time on a tour link it causes the associated values in the travel speed matrices to be temporarily incorrect.

Therefore, the Re-Ordering algorithm uses the delay time values and the static travel speed files to create dynamic travel speed matrices for that moment.

The methodology will proceed as follows:

Step 1: The time period will encompass a workday of 7:00am to 6:00p.m., divided into three time intervals ((7-10a.m.), (10a.m.-3:30p.m.), (3:30-6p.m.)), and travel time will be based on a step function.

Step 2: Select 10 randomly located nodes to visit, starting and ending at the depot, in the transportation network.

Step 3: Compute the shortest path for each of the 10 nodes, plus the depot, to all the other nodes for each of the 3 time intervals. These shortest path values will be used to create the travel speed matrices for each of the 3 time intervals. The actual shortest path geometry will be stored in a feature class called *TourLinks*#. The # will designate which time period the shortest path is form, ranging from 1 to 3. These feature classes will be used in Step 6. This will be a tight-coupled integration strategy and accessed through a custom button control, called *BuildMatrices*, in ArcInfo 8.

Step 4: Following the OAR rule of computing a complete tour first. The tour is computed by the restricted dynamic programming heuristics for the TDTSP (Malandraki and Dial 1996). The tour will be constructed based on the link travel times stored in the 3 travel speed matrices created in Step 3 with a start time at 7a.m. and an end time at 6p.m. This will be a hybrid-coupled integration strategy.
Figure 3.4 is a visual flowchart of the procedures in Steps 1 through 4 and Figure 3.5 is a visual flowchart of the procedures in Steps 5 through 7.

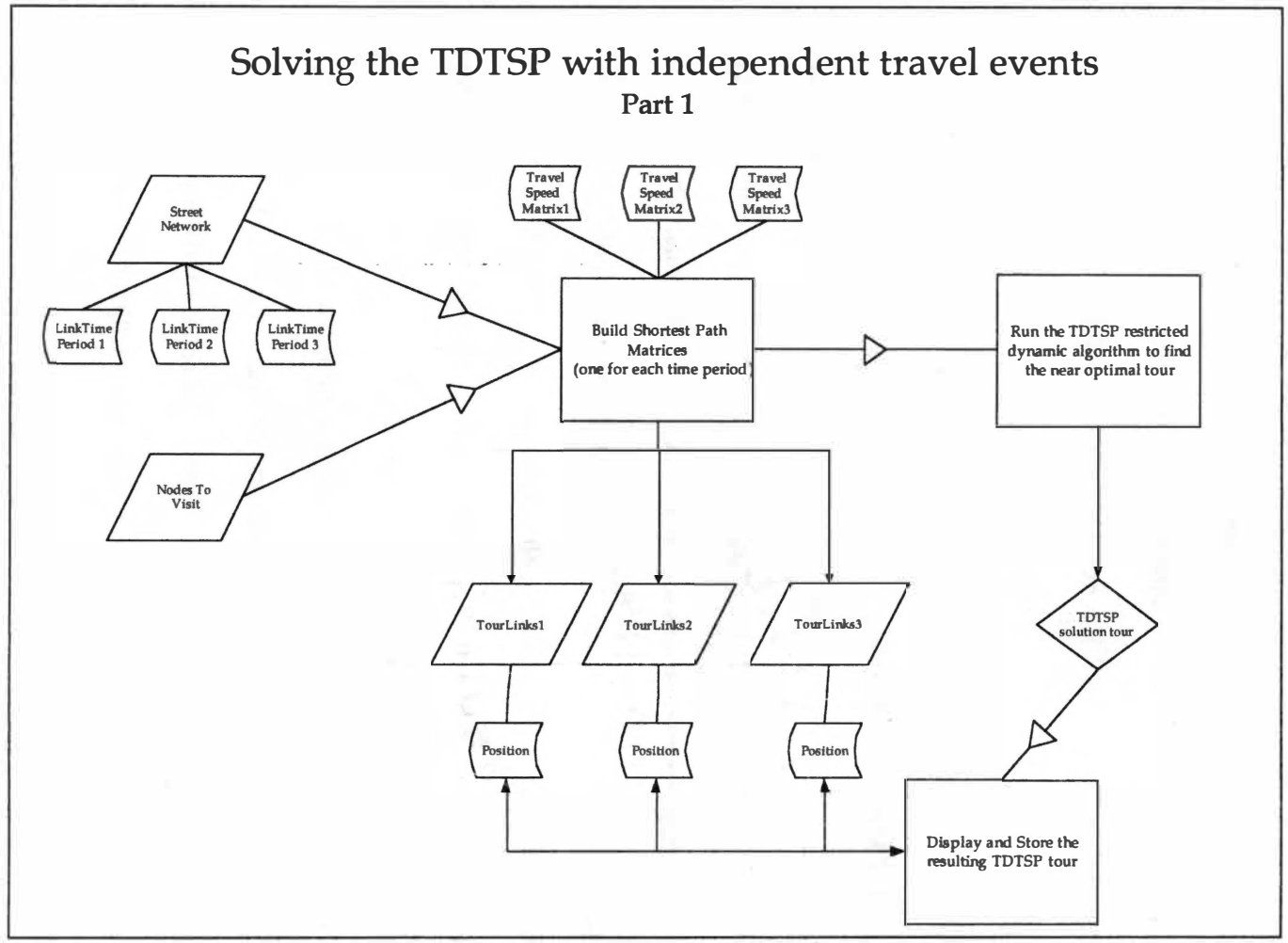


Figure 3.4 Flow chart of Steps 1 through 4.

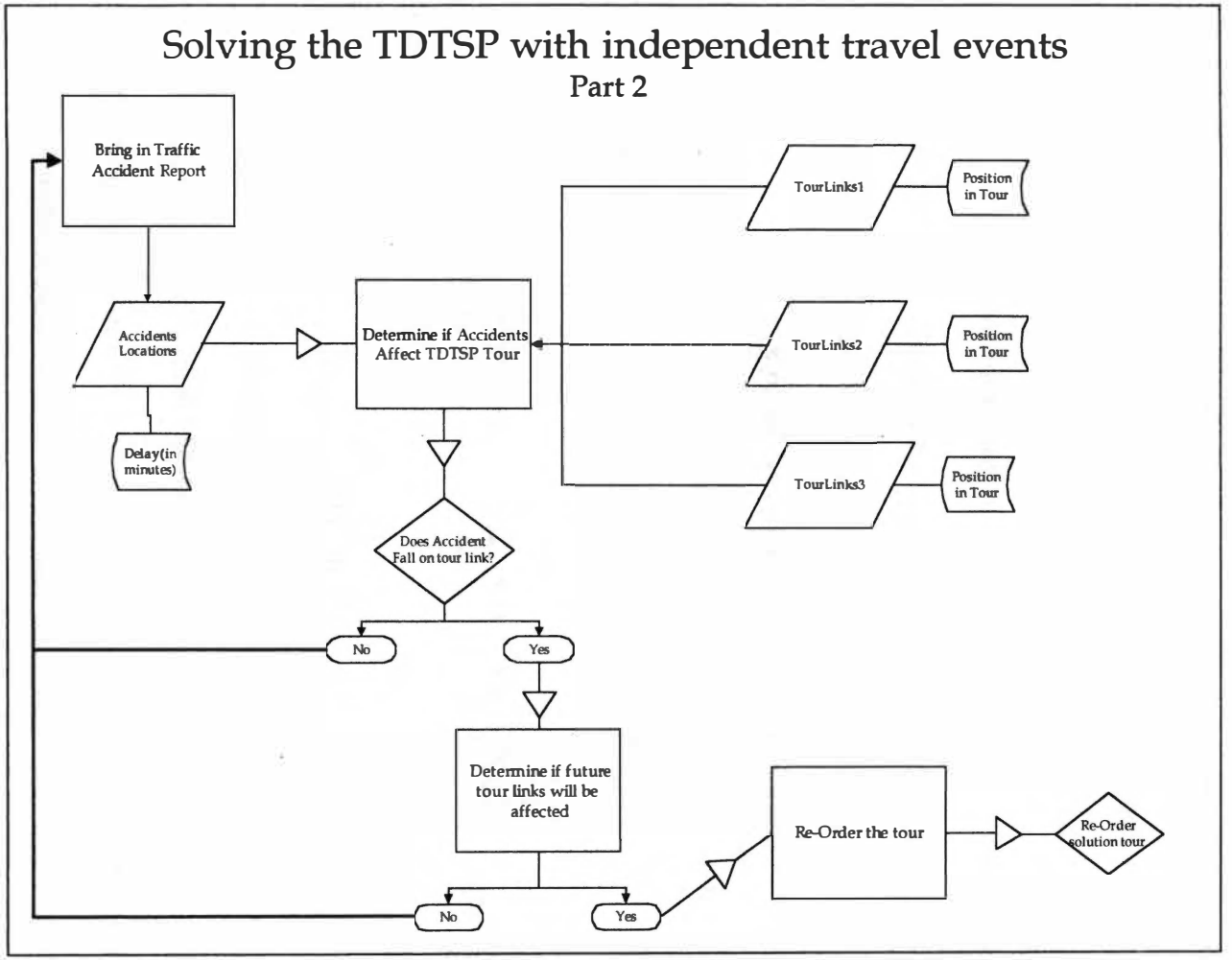


Figure 3.5 Flow chart of Steps 5 through 7.

Step 5: Assume that traffic reports are received hourly concerning vehicle accidents and the links directly affected with their delayed travel time speeds.

Step 6: Determine if Re-Ordering is necessary for the tour. All Re-Ordering decisions are made at stops. If the salesman is currently enroute to a stop, then the re-ordering decision will not be considered until he reaches that stop. Re-Ordering will occur if the links affected by the accident events are part of the tour, and these links of the tour have not already been visited. If so, proceed to Step 7. If not, then return to Step 5. Repeat till the time period is over. This will be a tight-coupled integration strategy accessed through a custom button control in ArcInfo 8.

Step 7: Run the Re-Ordering procedure to determine new tour. This will be a hybrid-coupled integration strategy.

3.3 Assumptions

My GIS implementation of the TDTSP will have certain assumptions to maintain tractability for the project. I will assume the street network to be symmetric, meaning that the cost of going from i to j in time period t is the same as going from j to i in the same time period t. Symmetric TDTSP is a simplification of the problem. According to Lawler et al. (1985), the asymmetric TSP is a *NP*-hard problem and is very difficult even to construct. To solve the asymmetric TSP requires heuristics where for the symmetric TSP, there are known exact algorithms for small sample sizes, such as Picard and Queyranne (1978) and Fox, Gavish and Graves (1980).

Since the focus of this thesis is on time-dependent and time-independent traffic events affecting the TDTSP tour, I will ignore asymmetric networks.

Another assumption will be the method by which I index time periods. I will be following Malandraki and Dial's (1996), and Hill and Benton's (1992) methodology of indexing by nodes. The time period t that the arc (i,j) falls into is determined by the accumulated time at origin node i. I assume that the time period t is the same for traveling the entire length of arc (i,j). The other option is indexing by links, as in the work of Ichoua, Gendreau and Potvin (2002). They consider the situation of where a link crosses more than one time period, which will cause variation in speed along that link. This requires much more computational work to manage the time periods indexed by links by the routing algorithm. Ichoua, Gendreau and Potvin used tabu search heuristics to try and handle the additional complexity of the problem. The tabu search results only guarantee a local-optimal solution.

3.4 Time Period and Step Function

The time period will encompass a workday of 7:00am to 6:00pm divided into three time intervals, one for morning rush hour (7-10), the second for in-between rush hour (10-3:30), and the third for evening rush hour (3:30-6). These periods are based on Gerald Barber's (1995) temporal distribution of trips through a typical weekday. Barber states that the amount of volume occurring during the peak periods at morning and evening rush hour are twice that at non-peak periods, causing the characteristic

congestion patterns experienced in majority of large cities in the U.S. The National Public Transportation Survey (NPTS) results from 1995 showed that 46.3% of trips occur not during the commuting periods, but actually from 9 a.m. to 4 p.m. These trips objectives were mainly social/family-related or work-related, for example going to the dentist or perhaps to a business lunch, and were relatively much shorter compared to the commute trips. Barber noted that the work commute only makes up about 15% of all trips, but argues that the geographical nonclustering of employment centers is a major contributor to these extreme congestion patterns. Another contributor is a widely dispersed employment base. The NPTS study supported this argument, noting commuting distances increased by 36.5% from 1983 to 1995. These observations make the case for using a time step function. The time step function I use in my project is based on Malandraki and Daskin's (1992) formulation $C_{ij}(t_i)$, wherein the cost of link (i,j) is denoted by C_{ij} and is a function of time t_i.

A geometric network is required to solve network routing problems in ArcInfo 8. The geometric network for my thesis is based on the *DCstreets* feature class. Therefore, the network weights used to solve the shortest path problem will be stored in *DCstreets*. The travel speeds for this network feature class will be stored in the attribute fields 'TimePd1', 'TimePd2', and 'TimePd3'. 'TimePd1' will reference the time interval from 7:00am to 10:00am, 'TimePd2' will store the travel speeds for the time interval from 10:00am to 3:30pm, and 'TimePd3' will store the travel speed cost from 3:30pm to 7:00pm. The values that are stored in these three fields are computed based on the length of the street links, stored in the 'Miles' attribute field of *DCstreets*, and the assumptions mentioned earlier that all one-way and two-lane

streets have speed limit of 30m/hr and all four-lane streets have a speed limit of 55m/hr. 'TimePd2' will be assumed the free flow state of traffic condition, while 'TimePd1' and 'TimePd3' will be assumed the congested peak periods and will be twice the value of 'TimePd2'.

3.5 Shortest Path Routine

ESRI's ArcInfo 8 will incorporate the TDTSP with stochastic travel events algorithm through a custom developed routing module I developed. The problem will be solved for 10 customers who are embedded within a transportation network. The GIS will handle the spatial features. Within the GIS, the nodes to visit (customers) will be designated on the transportation network. This is a true distance approach instead of a straight-line distance approach based on coordinates in a plane (Keenan 1998). Since TSP problems are commonly solved on directly connected graphs, a "pseudo-network" (see Figure 3.1) will be built from shortest path routines on the original street network (Miller and Shaw, 2001). A shortest path routine will be run for each one of the nodes to be visited, to all the other nodes to be visited during each time period. Each cost of this shortest path will be placed in its appropriate time period matrix, and these matrices will become the travel speeds used by the TDTSP heuristic algorithm. The resulting tour will then be displayed visually in the GIS. Tables 3.1 and 3.2 are examples of what a travel speed matrix for two time periods would look like for a network of 4 nodes plus the depot. Table 3.2 represents a more

Table 3.1 Travel speed matrix for time period 1.

Time Pd1	Depot	A	B	C	D
Depot	n/a	5	18	8	15
A	5	n/a	5	6	10
B	18	5	n/a	3	9
C	8	6	3	n/a	11
D	15	10	9	11	n/a

Table 3.2 Travel speed matrix for time period 2.

Time Pd2	Depot	A	B	C	D
Depot	n/a	8	24	12	19
A	8	n/a	9	10	14
B	24	9	n/a	7	12
C	12	10	7	n/a	15
D	19	14	12	15	n/a

congested time period than Table 3.1. The "pseudo-network" representations of these matrices are shown later on in this chapter in Figures 3.8 and 3.9.

The shortest path routine will be run in the GIS environment of ArcInfo 8, which has the geometric features to handle the network's topology. The custom control to run the shortest path calculation for the TDTSP matrices is written in VBA using ArcObjects, and calls the sample COM ActiveX DLL, *PathFinder*. The purpose of ESRI's COM object *PathFinder* is to demonstrate the ArcObjects necessary to perform shortest path routines between two or more points on a geometric network feature class. The *PathFinder* sample in its original format is

implemented via a custom *UIControlTool* from within ArcMap. Once the DLL is registered, the user adds code in the VBA editor in ArcMap to handle the events of this custom shortest path tool. The tool's events include: when the user selects the tool (a Select event), when the user performs a MouseDown event (a single click of the mouse), and when the user double clicks the mouse (a DblClick event). The Select event initializes the *PathFinder* COM object and provides the connection protocol between ArcMap and the *PathFinder* COM object. It does this by calling the method *OpenFeatureDatasetNetwork*, which takes the parameter of a *FeatureDataset* ArcObject. The MouseDown event 'catches' the points the user is selecting on the Map Document that the shortest path routine will be solved for and passes them to the *PathFinder* DLL through the *AddPoint* set property. The DblClick event signals that all points for which to solve the shortest path have been selected and calls *PathFinder*'s method *SolvePath* to solve the shortest path. *SolvePath* takes the parameter of the network weight, which for this *UIControlTool* I hard coded the attribute field 'Miles' of the network feature class *DCstreets,* to use as an example for the cost of the street links. This is not an accurate measurement for travel time and is only used to demonstrate this tool control. The same event then calls *PathFinder*'s method *PathPolyline* to get the cartographic geometry of the solution path. It is then displayed using methods from the ArcObjects *ScreenDisplay* interface. This particular COM sample can be downloaded from http://arconline.esri.com/ArcObjectsOnline/, and then navigating to the Samples/Network/PathFinder folder; or from ArcObjects Developer Help by typing keyword *PathFinder*. This ActiveX COM object is written in the three programming

languages that support developing component objects- Visual Basic (VB), Visual C++, and Delphi, and for this study the VB version was used.

A custom button control, called *BuildMatrices*, builds the travel speed matrices from the shortest path calculations by implementing methods from the *PathFinder* COM object (see Appendix 3 for script). The point feature class containing the depot node and the other 10 nodes to visit is created prior to invoking the *BuildMatrices* custom button. For this study, the feature class is called *Nodes-to-Visit*. *BuildMatrices* Click event is fired when the user selects it, and starts by initializing the *PathFinder* object and providing the communication protocol by calling the *OpenFeatureDatasetNetwork* method. *BuildMatrices* then proceeds to run a routine written in VBA that loops through the *Nodes-to-Visit* feature class and calls the *PathFinder* method *SolvePath* to construct the shortest path for each node to every other node for each of the 3 time periods. The three network weights needed are taken from the network feature class, *DCstreets*, and are listed under the attribute fields 'TimePd1', 'TimePd2', and 'TimePd3'. The routine takes the x,y coordinates of each of the nodes, stored in the feature's geometry property, and passes them to the *PathFinder* DLL to use as the points to calculate the shortest path. The costs of these shortest paths will make up the travel time matrices and are calculated on the network weight of travel time (in minutes) to traverse each link *(i,j)* for each of the 3 time intervals. The matrices are outputted to 3 text files: tsMatrix1, tsMatrix2 and tsMatrix3, which will be used by the TDTSP heuristic algorithm and the Re-Ordering algorithm. These output text files will contain the travel speeds on links when there is no independent travel event affecting the speeds. The geometry of the computed

shortest paths, obtained from the method *PathPolyline*, is converted to a new feature and is stored in a feature class named *TourLinks1*, *TourLinks2*, or *TourLinks3*- depending on the time period. Since I am indexing by nodes, I assume that all shortest path calculations to build the travel speed matrices occur during one time period. These feature classes are hard coded into the routine for this study. At the same time the shortest path geometry is being stored as a feature, the features associated attribute fields 'MyID' and 'To_Node' are filled and saved. These feature classes will be useful when determining if accidents (stochastic travel events) affect the TDTSP tour solution.

3.6 TDTSP Heuristic

The TDTSP will be solved using Malandraki and Dial's (1996) restricted dynamic programming heuristic algorithm. The restricted dynamic programming heuristic algorithm is solved on a directed graph $G(V,E)$, where V is the set of vertices (nodes) and E is the set of edges (links) that make up the graph G. A directed link is represented by an ordered pair of nodes (i,j), where i is the origin node and j is the destination node. My study concentrates on implementing a TDTSP problem with 10 stops. The graph is assumed complete and a 11x11 (including depot) time dependent matrix $C(t) = [C_{ij}(t_i)]$ is given for the travel times on each link (i,j) that is in E, where $C_{ij}(t_i)$ is a function of departure time t_i from node i. Malandraki and Dial (1996) use the restricted part of the algorithm for preventing the exponential explosion of time and storage requirements associated with dynamic programming (DP). The algorithm

is modified to only retain the *H* most cost efficient subtours at each stage of adding a new unreached node to each of the subtours. *H* in this case is equal to a 100, which Malandraki and Dial (1996) found to improve the solution by 13.8% on average. This is compared to having *H* equal 1 wherein, the problem is reduced to the classic Nearest-Neighbor method that generally does not produce optimal solutions.

H, the possible number of subtour variations saved, is stored in an array of 100 singly-linked list data structures, one list structure per subtour, called *PrevTour*[]. This singly-linked list data structure is actually an ATL class called *Tour* (see Figure 3.6), which not only saves the node sequence and cost of each of the subtours, but also provides other methods that will be explained below. When an unreached node is to be added, causing the subtours to increase its tour length by 1, an array of another 100 *Tour* classes, called *PotentTours*[], is used to store and compare the potential subtour variations encountered at that particular iteration of the TDTSP heuristic. The first part of the heuristic algorithm deals with what I call the Initialization Phase. An initializing array of only 10 *Tour* classes, called *SubTour*[], is filled with the node sequence of the depot, node 0, and another node to visit next. The array size of *SubTour* is only 10, since this is the number of possible variations for tours with 2 stops. Each *Tour* element in *SubTour* is pulled out individually to run through (a for loop procedure) the remaining nodes unreached, or in other words the nodes not yet reached and not stored in *SubTour*. The unreached nodes for each *SubTour* are stored in an array called *Marked*[],where 1 is used if the node is in the tour and 0 otherwise. After an unreached node has been added to the subtour, it receives a value of 1 in the *Marked* array. *Tour* class supplies a function,

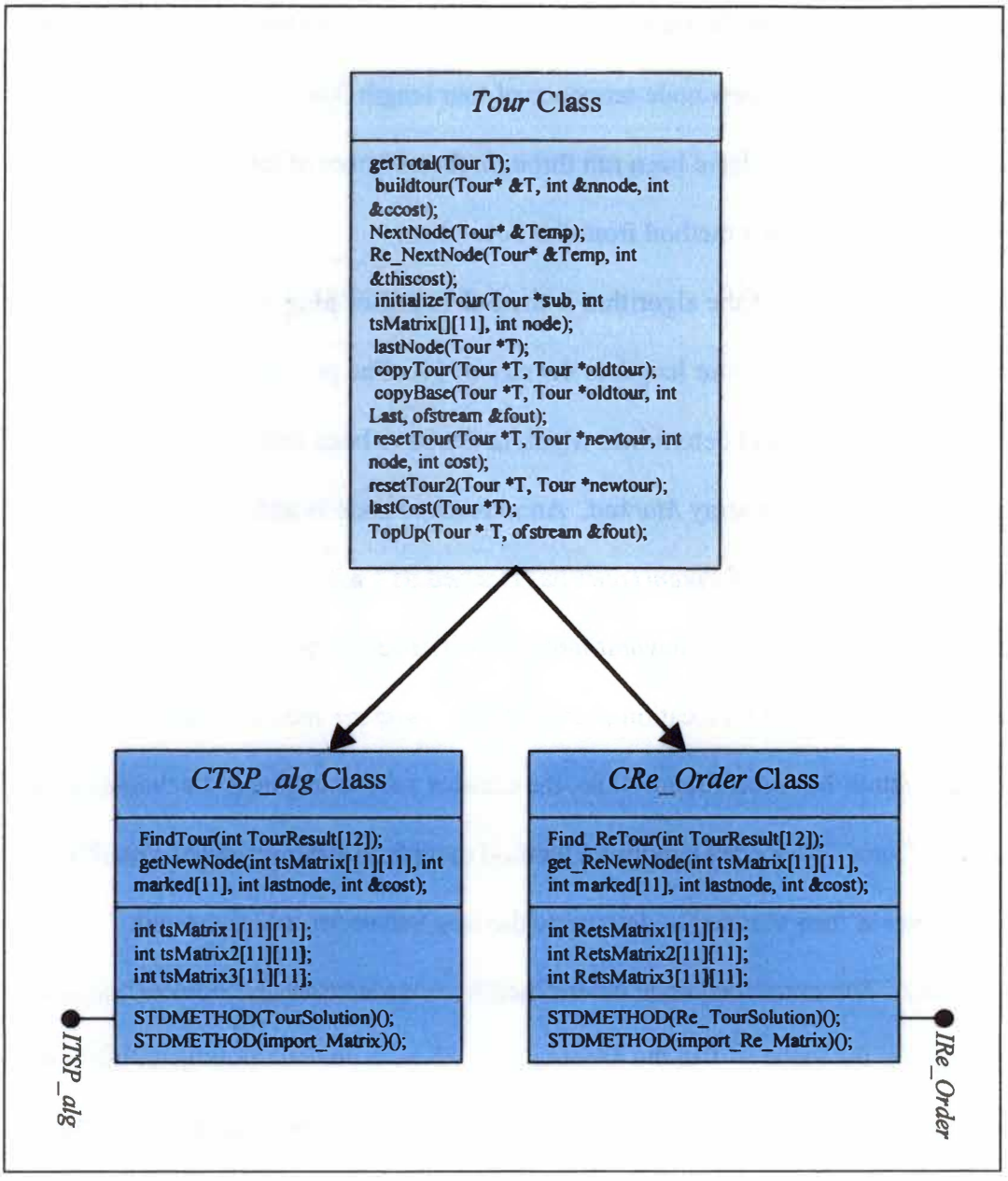


Figure 3.6 My custom ATL COM heuristic object model.

get_NewNode, which determines the next unreached node to add to the subtour as well as which tsMatrix to use for the cost of adding the node to the subtour. Recall that the tsMatrices store the travel speed costs of all the links that can be used by the TDTSP solution. The new node sequence of tour length 3 is stored in *PotentTours,* and when all *SubTours* have been run through, the subtours of length 3 are copied into *PrevTour* through a method from the *Tour* class.

The next part of the algorithm is the SubTour Building Phase and runs (for loops through) until the tour length is the size of 11. The procedure starts at the first subtour in *PrevTour* and determines which nodes have been and have not been added to the subtour from the array *Marked.* An unreached node is added to the subtour and stored in *PotentTours.* If *PotentTours* has reached its limit of 100 subtours, two variables called *worstcost* and *worstindex* are initialized. These variables store the worst cost subtour and its location in *PotentTours,* and are used to determine if a cheaper subtour has been found. If so, the cheaper subtour replaces the worst subtour in *PotentTours. Tour* class supplies a method to replace subtours called *ResetTour. PotentTours* is then scanned to determine the new values for *worstcost* and *worstindex.* The unreached node determined from the subtour in the *PrevTour* array is then given the value of 1 in the *Marked* array so as to prevent looking at this node again during this iteration of the heuristic. This is repeated until all unreached nodes for that *PrevTour* subtour have been examined for their potential of being placed in *PotentTours.* The procedure then moves to the next subtour in *PrevTour* and repeats the steps above until all subtours in *PrevTour* have been examined. When the last subtour has been examined, the subtours in *PotentTours* are copied to *PrevTour* and

the whole process starts over until all nodes, in this case 10, have been added to each of the subtours. The last step is to go from the last node in subtours back to the depot and the cheapest tour is then selected, from the set of *H*. The code for this heuristic can be found in Appendix 1. Figure 3.7 offers a flow chart visualization of this process as well as an example is presented in section 3.6.2.

3.6.1 ArcInfo 8 Implementation of the TDTSP

The Active Template Library (ATL) COM object I created to implement this heuristic allows the user to access the algorithm through the object's interface, *ITSP_alg*, with its methods *import_Matrix* and *TourSolution* (see Figure 3.6). The actual project can be viewed by using the CD located in the back pocket of this thesis. The method *import_Matrix* reads in the travel speed matrices needed to solve the TDTSP. The method *TourSolution* implements the TDTSP heuristic by making calls to the class this interface points to, called *CTSP_alg*. The function *FindTour*, is called by the interface method *TourSolution* and implements the procedures of the heuristic algorithm. As terminology can be confusing, I will attempt to clarify. Minimally COM objects have both an interface and a class. When the COM objects are called in a VBA environment in ArcMap, the only calls available to the user are the methods and properties offered by the interface(s). Hidden from the user are the class' functions and variables, which are internally accessed by the methods and properties of the interface(s).

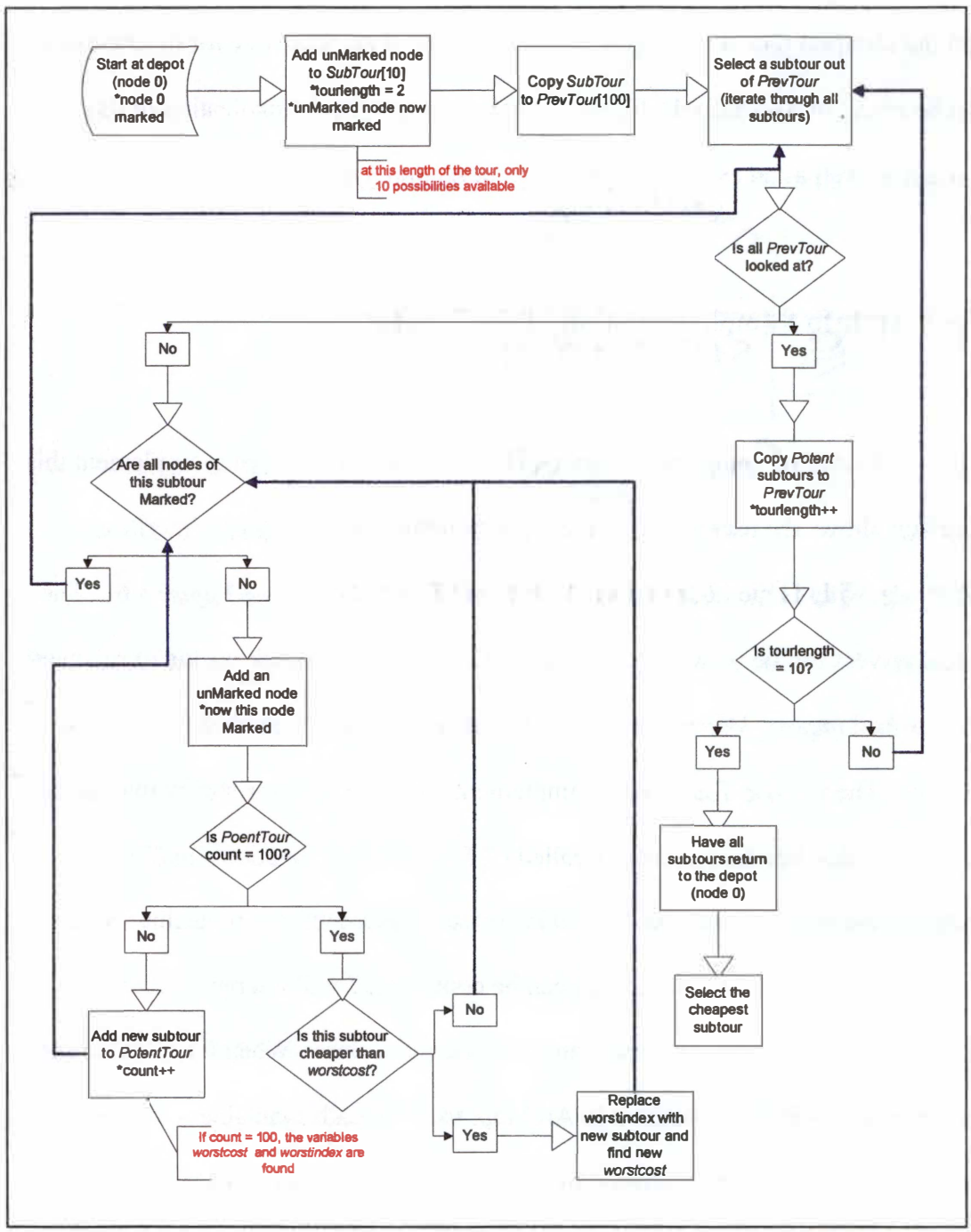


Figure 3.7 Flow chart of the process of the TDTSP heuristic.

This thesis uses a hybrid-couple integration with ArcInfo 8. The hybrid coupling is a combination of techniques from loose-coupling and tight-coupling strategies. To run the TDTSP algorithm, the user must execute the program from a custom *UIButtonControl*, named *RunTDTSP,* from within ArcInfo 8. The COM object is an ATL project and the resulting DLL is registered with ArcInfo. Prior to selecting this custom button, the DLL must be registered and the reference to *TDTSPLib,* my COM type library, set in the VBA editor. The button then instances the object interface called *TDTSPLib.Tsp_alg.* This interface allows for calls from VBA to solve the TDTSP problem. This TDTSP algorithm stores its solution in a text file called TDTSPtour. The user can then display the tour results onscreen and store the position number of each tour link in its appropriate *TourLinks#* feature class table by engaging the custom *DisplayTour* button control (see Appendix 3 for script). *DisplayTour* reads in the text file TDTSPtour through a for loop procedure. The routine reads in the first node (*node1*), the second node (*node2*) and cost. The routine determines which feature class *TourLinks#* the link (*node1,node2*) is from and the specific feature that stores that link is found using the variables *node1* and *node2.* The corresponding attribute field, 'Position', stores the link position value. This provides the user with information about the order in which the nodes are visited. This segment of the TDTSP tour solution is displayed on the screen. *Node1* is given the value of *node2*, and the next node (*node2*), and cost are read in from the file TDTSPtour. These steps continue until all links have been read in and stored.

3.6.2 TDTSP Example

To illustrate the TDTSP restricted dynamic programming heuristic please review the following Figures 3.8 through 3.13 for an example using 4 nodes plus the depot. For this demonstration of the heuristic we will assume only 2 time periods and that the number of subtours to store by the restricted dynamic programming algorithm, *H*, will be 2. The red node represents the depot, and time period 1 will be from start time 0 to 10, with period 2 starting at time 11. Figure 3.8 is the original "pseudo-network" to be used to solve the TDTSP for time period 1, while Figure 3.9 is the "pseudo-network" for time period 2 and represents a more congested period of travel. Remember that one link in the "pseudo-network" can be comprised of more than one link in the original street network and Figures 3.8 and 3.9 are an abstraction of the real network. These "pseudo-networks" are also the visual representation of the travel speed matrices, one per time period. The travel speeds matrices for this example are displayed in Table 3.1 and Table 3.2 presented previously.

The first step is to add the depot to the subtours array *PotentTours*, which has a size of 2 for this example, and the accumulated time equals 0, since we have not traveled anywhere. The depot becomes a marked node not to be considered again in the array *Marked.* From the depot, all the remaining unmarked nodes are viewed having the potential of being stored in the subtour *PotentTours* array. The time period to use depends on the accumulated time at the origin node, in this case the depot node with a time of 0. For example, subtour 1 in the *PotentTours* array would

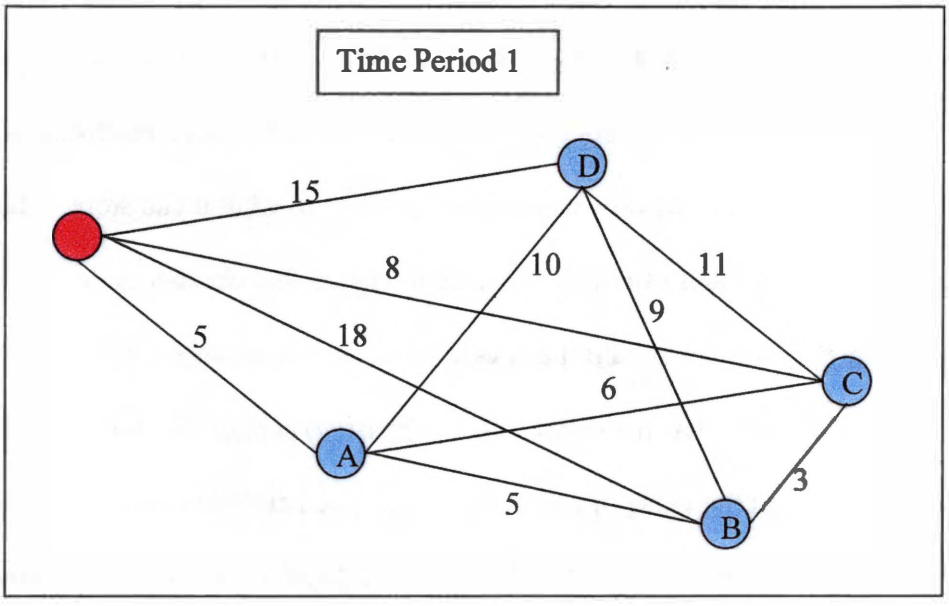


Figure 3.8 Generalized view of the "pseudo-network" for time period 1.

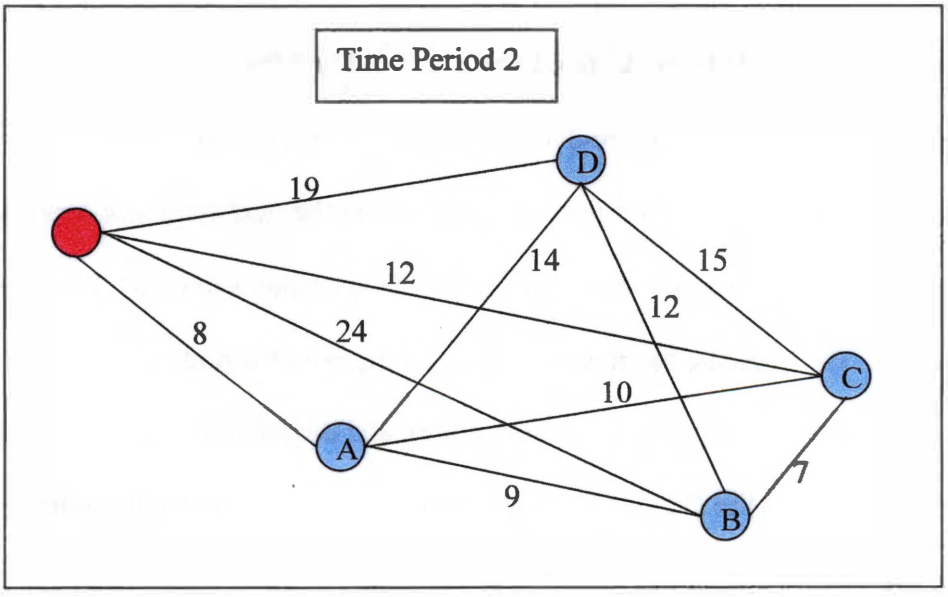


Figure 3.9 Generalized view of the "pseudo-network" for time period 2.

start with the sequence depot-A, and A becomes temporarily marked in the array *Marked*. The first time period will be used to get the travel time cost, since the accumulated travel time at the origin node, the depot, is 0. Then the next iteration selects an unmarked node for subtour 2, depot-B, and B has become temporarily marked. The subtour array now has reached its limit to what it can store. The variables *worstindex* and *worstcost* are used to ensure that cheaper node sequences are saved in *PotentTours* should they occur later. In this example, subtour 2, comprised of depot-B, has the worst (most expensive) cost of 18. So, the value of *worstindex* is 2 and the value of *worstcost* is 18. The heuristic continues to look at the remaining unmarked nodes. For the sequence depot-C, the travel time cost is 8 and is cheaper than *worstcost*. Therefore, using these variables, we know to replace subtour 2, the index value stored in *worstindex*, in the *PotentTours* array with the sequence depot-C. C becomes temporarily labeled and the values stored at *worstindex* and *worstcost* are re-determined, which are now the cost of 8 and the index of 2. The heuristic continues on and looks at the next sequence depot-D, which has a travel time cost of 15. Since this is not cheaper than *worstcost*, this sequence is discarded. The heuristic has now looked at all the possible node combinations for tour length of 2 (the depot plus the next node), and the subtours stored in *PotentTours* array are copied to the *PrevTour* array. Figure 3.10 shows the results stored in *PrevTour* after the first iteration for tour length 2.

The *Marked* array is now re-initialized to values of 0. The next iteration takes the first subtour stored in *PrevTour*, depot-A, and marks the depot and A in the *Marked* array so it will not be considered as a nodes to add to the subtour of tour

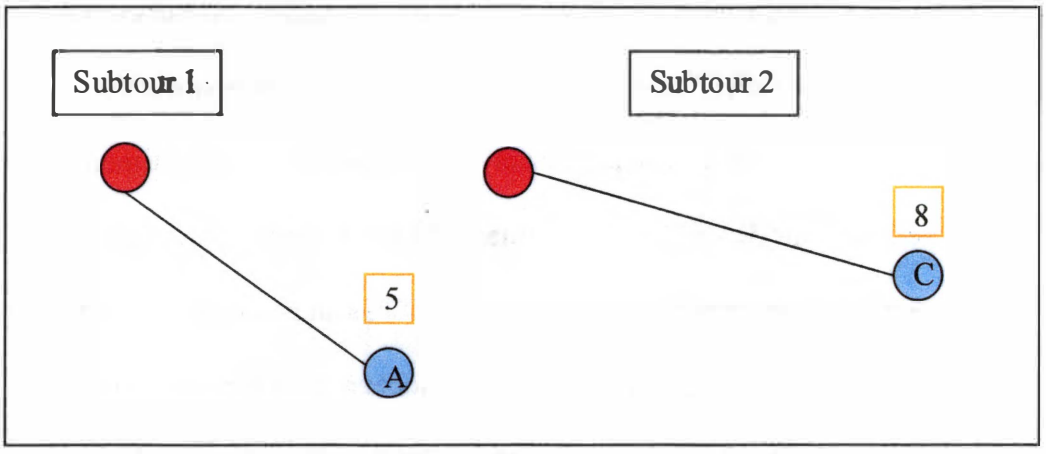


Figure 3.10 The subtours saved in *PrevTour* for tour length 2.

length 3. Since, the accumulated travel time cost for the first subtour in *PrevTour* is 5, the time period used is still 1. The heuristic then looks for the first unmarked node, which is node B. *PotentTours* subtour array is re-initialized with the first subtour in *PotentTours* being depot-A-B with the accumulated travel cost of 10. Next, *PotentTours* subtour 2 is filled with the node sequence depot-A-C, with the time of 11. Since, the number of subtours to store has reached the limit of 2, the variable *worstindex* is assigned the *PotentTours* subtour index of 2 and *worstcost* is assigned the time of 11. The next sequence reviewed by the heuristic is depot-A-D with a cost of 15. This is still more expensive than *worstcost* so depot-A-D is discarded. The heuristic has looked at all the combinations of unmarked nodes that can be added to *PrevTour* subtour 1 sequence of depot-A and therefore, the heuristic moves on the *PrevTour* subtour 2 sequence of depot-C. Since, the accumulated travel time is 8, the travel cost will be based on time period 1. The *Marked* array is again re-initialized and the only nodes that are now marked are the depot and C. The first unmarked node to be added to depot-C is A, with the accumulated cost being 14. This is still

not cheaper than *worstcost* so this subtour sequence is discarded. Sequence depot-C-B has the cost of 11, which is the same as *worstcost*. The heuristic works by changing the subtour stored at *worstindex* only if the *worstcost* is greater then a new sequence of nodes, thus depot-C-B is discarded. This is a heuristic procedure and does not guarantee an optimal solution. Continuing on, sequence depot-C-D has the cost of 19 and is also discarded. The heuristic has looked at all the possible node combinations for tour length of 3 and once again, the subtours stored in *PotentTours* array are copied to the *PrevTour* array. Figure 3.11 shows the results stored in *PrevTour* after the second iteration for tour length 3.

Next, the heuristic iterates through the process again for tour length of 4. It takes the first subtour stored in *PrevTour*, which is depot-A-B, and marks the depot, A, and B, so they will not be consider as nodes to add to the subtour of tour length 4. Since, the accumulated travel time cost for the first subtour in *PrevTour* is 10, the time period is still 1. The heuristic then looks for the first unmarked node, which is node C. *PotentTours* subtour array is re-initialized with the first subtour in

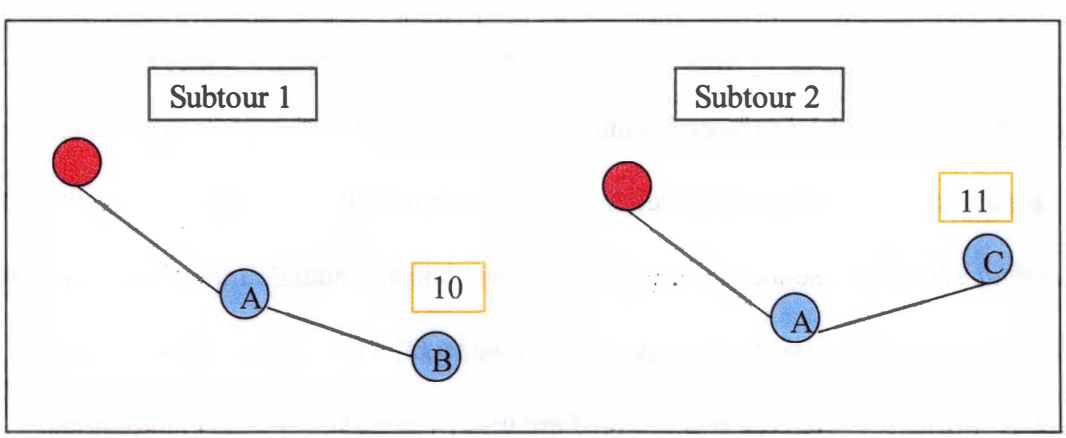


Figure 3.11 The subtours saved in *PrevTour* for tour length 3.

PotentTours being depot-A-B-C with the accumulated travel cost of 13. Next, *PotentTours* subtour 2 is filled with the node sequence depot-A-B-D, with the time of 19. Again, the number of subtours to store has reached the limit of 2, the variable *worstindex* is assigned the *PotentTours* subtour index of 2 since it has the *worstcost.* This demonstrates one of the downsides of indexing time periods by nodes. Obviously, the sequence depot-A-B is right on the cusp of the next time period. However, since my algorithm determines which time period to use by the accumulated time at the origin node, the time cost will be from the first time period though the links, in reality, will most likely cross two time periods. The computational complexity of indexing by links is not handled for this study but is a very relevant future consideration for making the TDTSP solution a more realistic one. The heuristic has looked at all the combinations of unmarked nodes that can be added to *PrevTour* subtour 1 sequence of depot-A-B, therefore, the heuristic moves on to the *PrevTour* subtour 2 sequence of depot-A-C. Since, the accumulated travel time is 11, the travel cost will be based on time period 2. The *Marked* array is re-initialized and the only nodes marked are the depot, A, and C. The first unmarked node to be added to depot-A-C is B, with the accumulated cost being 18. This is less than *worstcost* so the *PotentTours* subtour 2 is replaced with depot-A-C-B. The new values for worstindex and worstcost are determined, and the heuristic moves on. Sequence depot-A-C-D has the cost of 26, which is greater than the *worstcost,* and is discarded. The heuristic has looked at all the possible node combinations for tour length of 4. Despite having the problem of *PrevTour* subtour 1 using time period 1

cost, and *PrevTour* subtour 2 using time period 2 costs, *PrevTour* subtour 2 still had a node sequence that was cheaper than one from *PrevTour* subtour 1 for tour length of 4. One might assume that since time period 2 is the congested period and has inflated travel time compared to time period 1, that there would be no subtours selected from subtour 2 of *PrevTour*. Though this did not occur in this instance, it is still a problem to consider for future research. The subtours stored in *PotentTours* array are copied to the *PrevTour* array. Figure 3.12 shows the results stored in *PrevTour* for tour length of 4.

Both accumulated travel times in *PrevTour* subtours are now in time period 2. Since, there is only one unreached node left, node D, the heuristic automatically adds the last unreached node to the subtours in *PrevTour*. Subtour 1 sequence is now depot-A-B-C-D with a time of 28 and subtour 2 is depot-A-C-B-D with a cost of 30. The last node in this instance happens to be the last node for both subtours, which is not always the case. The last step is to return to the depot, and again the travel time

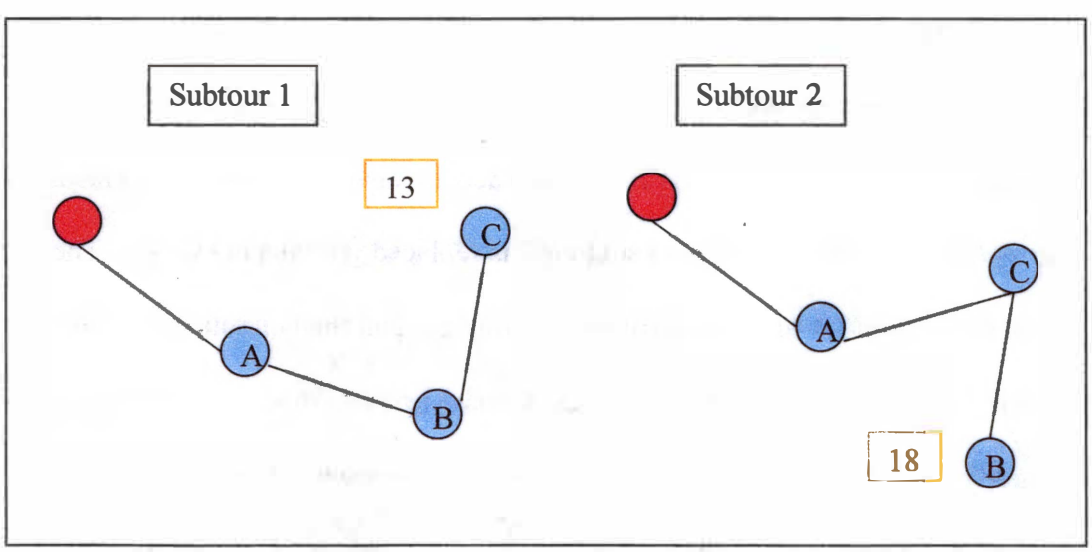


Figure 3.12 The subtours saved in *PrevTour* for tour length 4.

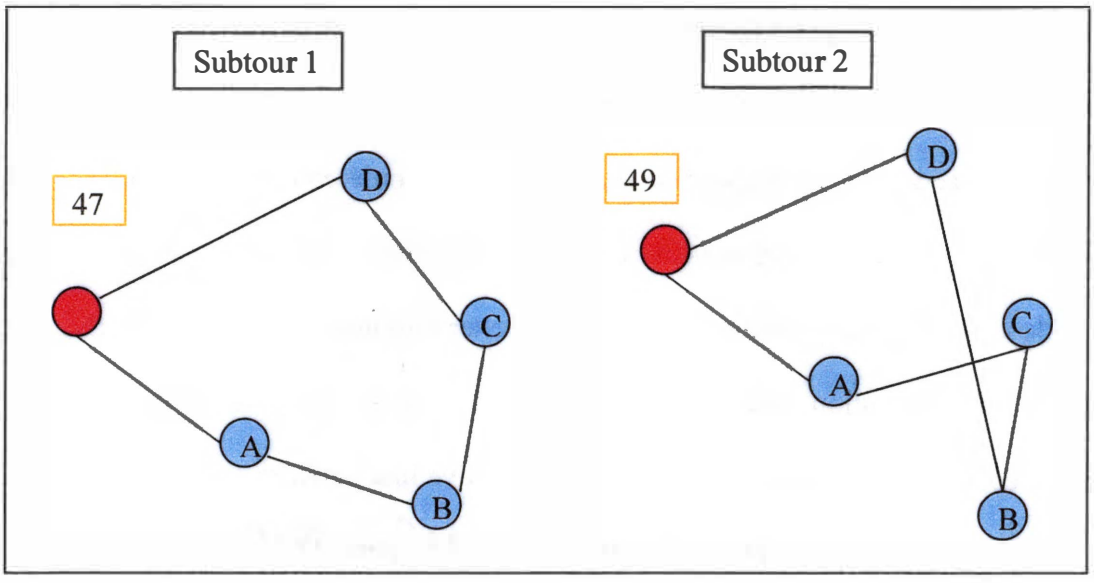


Figure 3.13 The final subtours saved in *PrevTour*.

period will be 2. Figure 3.13 shows the final results for *PrevTour* subtours. The only remaining step is to select the cheapest tour from among the saved subtours in *PrevTour*, which in this example is subtour 1 with an accumulated travel time of 47 for the TDTSP tour. In this example, the "pseudo-network" links cross in subtour 2, though this may not actually be the case when drawn on the base street network. The "pseudo-network" is drawn in a generalized style for visual simplicity.

3.7 Traffic Reports

Traffic reports are assumed to be available from an external source, such as an ITS, every hour. ITS are typically aimed at individual users, however this information is also pertinent to commercial users of the transportation network for solving routing problem. A degraded solution might result if a major vehicle accident

occurs, which directly affects one or more links that are used in a TSP tour. These accidents are stored in point feature classes named *Accidents_8am* through *Accidents_5pm*. To prevent the TSP tour from deteriorating into a lower quality solution, the nodes not yet reached on the tour can be re-ordered to provide an alternative sequence of nodes to visit. Currently, my thesis only evaluates whether the node sequence can be changed to provide a better tour than continuing with the original TDTSP solution, with traffic accidents adding delay cost(s) to the overall solution cost, instead of possibly finding an alternative link between *i* and *j*.

The first part in this process, after receiving the updated traffic reports, is to determine if the street arcs that have incurred a delay in travel speed (due to accidents) are used by the TDTSP tour. This is achieved by using the custom menu *Update* that contains the tools *Select By Attribute* and *Select By Location.* These selection tools are available in ArcInfo 8, and require no customization on my part. The first step is to use the *Select By Attribute* tool to identify the "pseudo -networks" links that are used by the TDTSP solution tour. These are stored in the feature classes *TourLinks1, TourLinks2,* and *TourLinks3* attribute field 'Position'. The second step uses the *Select By Location* tool to identify the "pseudo-network" link(s) directly affected from accident(s) downloaded from the traffic report. If the accident(s) occur on link(s) in the tour, the next step is to use the custom *MyUserUpdateForm* macro button to determine at which link the traveling salesman is currently located. The *MyUserUpdateForm* macro button, when clicked by the user, launches the custom VBA form, called TDTSP Update. The user then fills out the necessary information to run the *FindSelectedAccidents* method stored in the VBA class module *GetUpdates*

(see Figure 3.14 and Figure 4.8). This routine determines if the link(s) directly affected have not been reached by the tour. If there are link(s) that have not yet been reached and are affected by accidents, the user will be prompted with a message box stating that the Re-Ordering algorithm should be run to determine if a better sequencing of nodes is available. The *MyUserUpdateForm* macro outputs to a text file, called ReOrder, the necessary information to be used by the Re-Ordering algorithm. This information is comprised of the number of links affected, the position of the link(s) affected in the tour, the to-node for those link(s), the delay value (in minutes) of the affected link(s), which are stored in the *Accidents* feature class attribute field 'DelayValue', and the current time.

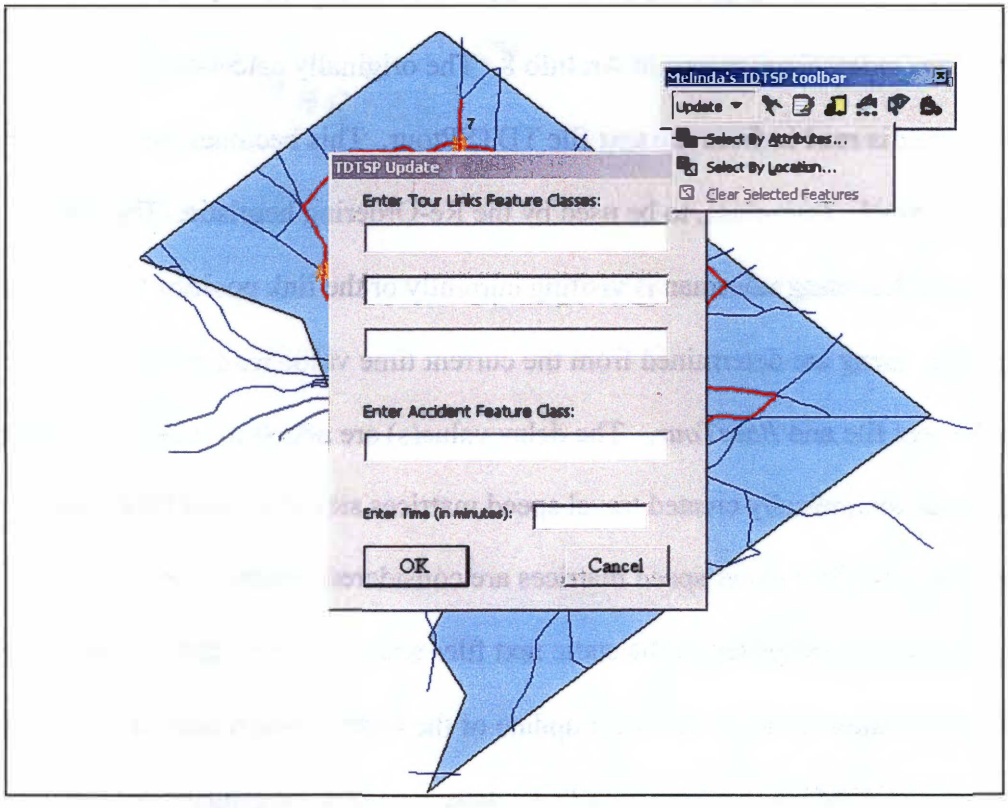


Figure 3.14 View of tools used for determining if Re-Ordering is necessary.

3.8 Re-Order Heuristic

The Re-Ordering procedure requires the same three originally calculated time matrices for the TDTSP heuristic; tsMatrix1, tsMatrix2 and tsMatrix3. These files store the shortest path costs for the three time periods when the street network is unaffected by stochastic events of traffic accidents. The Re-Ordering algorithm also needs the number of links affected (*NumOfLinks*), the delay cost of the affected tour link(s) due to updates from the traffic reports, the to-nodes of the affected links, and the current time. The Re-Ordering algorithm starts by looping through *NumOfLinks* to read in the to-node values and delay values stored in the file, ReOrder, outputted by *MyUserUpdateForm* macro in ArcInfo 8. The originally calculated TDTSP solution tour is read in from the text file TDTSPtour. This becomes the *BaseTour*, which is a single *Tour* class, to be used by the Re-Ordering heuristic. The node position the traveling salesman is visiting currently or the link position the salesman is currently using are determined from the current time value read in from the ReOrder text file and *BaseTour*. The delay value(s) are added to their corresponding place in the dynamically created travel speed matrices stored in the COM class *CReOrder_alg*. The travel speed matrices are considered dynamic, or 'on-the-fly', because they are comprised of the static text files and the current delay values for this particular moment in time. The next update of the traffic reports may have these links back to the originally calculated speeds, so these matrices are created only

temporarily. The tsMatrix to which the delay values belong, and their exact position, are determined from the to-node values and the original *BaseTour*.

The procedures for the Re-Ordering algorithm are very similar to the TDTSP restricted dynamic programming heuristic in computing the original solution tour (see Figure 3.7). The only difference is that the procedure starts from the current node, which will have a tour length greater than 1, instead of at the depot, which starts at a tour length of 1. The nodes yet unreached by the tour and the length of the tour are determined from the information in the ReOrder text file, and *PrevTour* is initialized with the nodes already reached and a new unreached node. *PrevTour* and *PotentTours* are still class *Tour* arrays of size 100. The procedure then continues by adding new nodes to the subtours determined in *PotentTours*, and then once again, when all the unreached nodes have been added, all the subtours return to the depot. The cheapest tour is selected and is compared to the original tour to see if a sequence change has occurred. If so, this tour becomes the new TDTSP tour with the cheapest tour being outputted to the text file TDTSPtour. The code for this heursitic can be found in Appendix 2.

3.8.1 ArcInfo 8 Implementation of Re-Order

The Active Template Library (ATL) COM object I created to implement this heuristic allows the user to access the algorithm through the object's interface, *IReOrder_alg*, methods *import_Re_Matrix* and *Re_TourSolution* (see Figure 3.6). The method *import_Re_Matrix* reads in the travel speed matrices previously

calculated for the TDTSP. The method *Re_TourSolution* implements the TDTSP heuristic by making calls to the class to which this interface points, called *CReOrder_alg*. The function *Find_ReTour*, is called by the interface method *Re_TourSolution,* and implements the procedures of the heuristic algorithm.

This chapter has presented the logic and structure of the TDTSP with stochastic events. Now that these have been discussed and the necessary software developed, the next chapter persents numerical results of the ArcInfo 8 based implementation.

Chapter 4

Results

This research provides tools to solve a TDTSP with stochastic travel events in an ArcInfo 8 GIS environment. Prior to this research, TSP problems had to be solved in another software, such as TransCAD. Using such software produces a solution that ignores the time-dependent component as well as the stochastic components. In TransCAD, manual intervention and excellent user knowledge are required if the user wishes to consider these components when solving this problem. To provide users with the functionality to solve the TDTSP with stochastic travel events within ArcInfo, I developed custom COM objects that are embedded within ArcMap. Using this strategy, I was able to take full advantage of the excellent visual tools for analysis and display provided by the GIS environment. Jones (1994) argued that the process of problem solving has different stages of visualization. My implementation trys to follow these visual steps through the identification of the original TDTSP problem, managing of the spatial data (the street network and stop locations), the development of the TDTSP model, and finally the display of the results, as well as re-validating the solution against accidents. The advantageous nature of the visual tools available can be illustrated through the GIS's ability to display the levels of service (LOS) for each link during each time period, and the TDTSP tour. The results visually demonstrate how the solution avoids severely congested areas. These visual tools are also

important for the stochastic component, in that the visual graphics allow the display of the links affected by accidents.

The first section of this chapter will outline the functions I developed to solve a TDTSP with stochastic events in ArcInfo 8. The second section of the chapter focuses on the effectiveness of the Re-Ordering of the TDTSP, as compared to the result of the non re-ordered TDTSP solution. The two overall travel cost results will be compared for 10 instances of the TDTSP problem and will be comprised of 10 randomly generated nodes to visit. The results show that the Re-Order strategy produces an alternative tour solution that returns lower overall travel cost than the TDTSP that does not Re-Order in the face of unpredictable changes on the network. This validates the consideration of time-stochastic travel events along with dependent events to provide more accurate variations in travel speed when solving the TSP problem for real world applications.

4.1 User Implementation of the TDTSP with Stochastic Events

I will demonstrate the custom spatial function and tools I have developed by going through the procedures, as would any user who is trying to solve the TDTSP with stochastic events. Prior to opening ArcMap, the required feature class and data need to be created or imported into a geodatabase. This problem requires a street network from which a geometric network is derived. Because this is a network optimization problem and its solution is derived on this network, the geometric

network is essential to perform trace path calculations. For this demonstration, the polyline feature class called *DCstreets* holds the arcs used to create the geometric network. The street network class must have three attribute fields that contain the travel speed for each arc for the 3 time periods. Also essential to defining the TDTSP problem is the location of the depot and the additional 10 nodes/stops to visit. The depot and stops are saved in a point feature class, which for this demonstration, is called *Nodes-To-Visit.* My design of this implementation approach also requires three new polyline feature classes that store the geometry of each shortest path calculation for all the possible links that can be used by the TDTSP tour (the "pseudo-network"). For this research, the feature class' names are hard coded into the VBA routine and they are named *TourLinks1, TourLinks2,* and *TourLinks3.* These feature class' attribute tables will store the unique ID for each shortest path ('MyID'), the to-node of this shortest path ('To_Node'), the position this link is used in the TDTSP solution tour ('Position'), and the accumulated travel time at the links used by the tour ('Acc_Time'). These feature classes are initially empty of features, but are filled with features after the invoking of the *BuildMatrices* button. Additionally, this custom button will also fill the attribute fields 'MyID' and 'To_Node'. The attribute fields 'Position' and 'Acc_Time' will have NULL values until executing the *DisplayTour* module. Once these feature classes and their required attribute fields have been created, ArcMap can be launched to create the map document containing the required feature classes (see Figure 4.1). Upon opening ArcMap, the user encounters the

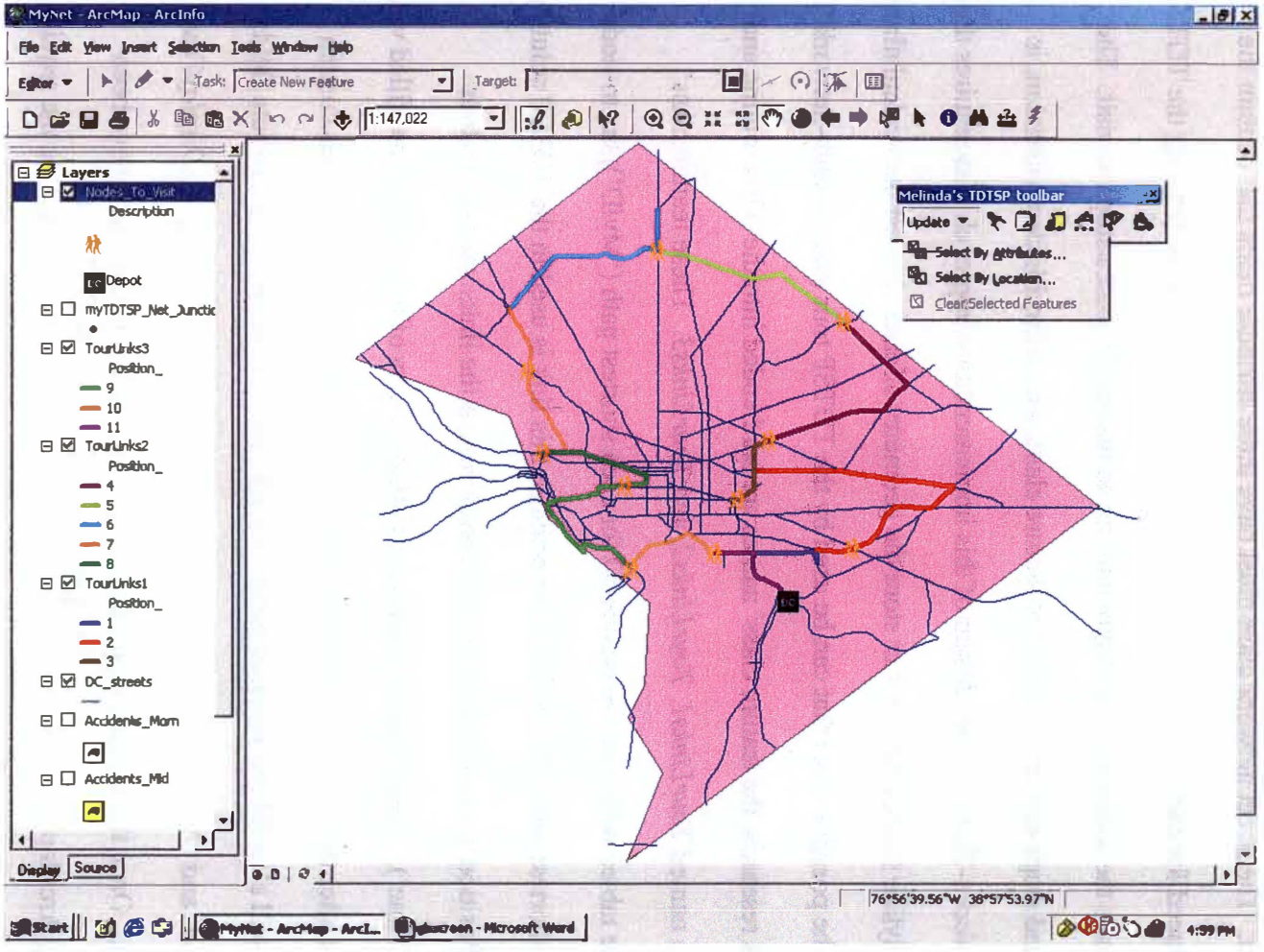


Figure 4.1 My TDTSP with independent events project view.

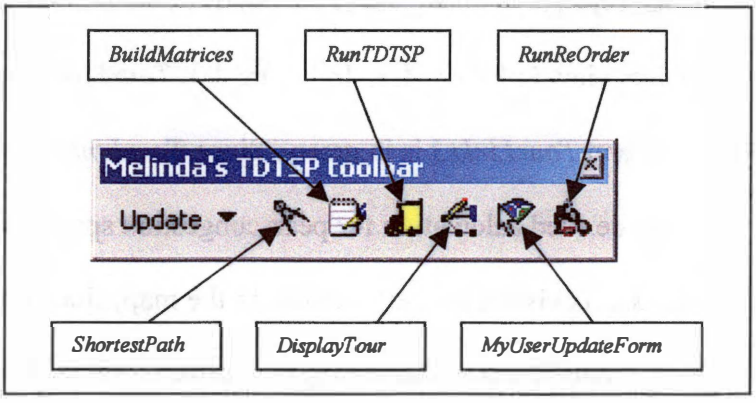


Figure 4.2 View of my toolbar with its custom tools and buttons.

custom toolbar, *Melinda's TDTSP toolbar*, which contains the custom tools to solve the TDTSP with stochastic travel events.

The first step to solving the TDTSP with stochastic events in ArcMap is to execute the *BuildMatrices* custom *UIButtonControl.* Figure 4.2 shows an enlargement of the custom toolbar *Melinda's TDTSP toolbar*, which provides a better overview of each custom button. The *BuildMatrices* custom button is used to build the travel speed matrices from the shortest path calculations by implementing methods from the *PathFinder* COM object. Clicking on this custom button runs a VBA routine that loops through the *Nodes-to-Visit* feature class, and calls the *PathFinder* method *SolvePath* to construct the shortest path for each node to every other node for each of the 3 time periods. The appropriate network weights are used, which are stored in *DCstreets's* attribute table in the fields 'TimePd1', 'TimePd2', and 'TimePd3'. The costs of these shortest paths will make up the travel time matrices that are outputted to the text files: tsMatrix1, tsMatrix2 and tsMatrix3, which are used in the TDTSP heuristic algorithm and the Re-Ordering algorithm. Each feature is created through the geometry of the computed shortest paths, obtained from the method *PathPolyline*,

and is stored in its appropriate feature class (*TourLinks1*, *TourLinks2*, or *TourLinks3*) as shown in the project view in Figure 4.3. In Figure 4.3, *TourLinks1* is in red, *TourLinks2* is in blue and *TourLinks3* is in green. Since *TourLinks1* and *TourLinks3* are based on the same derived calculation for peak congestion speeds, they overlap completely. *TourLinks2* is visible for certain links in the map, since this is based on travel time at free flow conditions. Also during the custom routine, the features associated with attribute fields 'MyID' and 'To_Node' are filled and saved as seen outlined in a red box in Figure 4.4.

The next step is to launch the TDTSP restricted dynamic programming heuristic (Malandraki and Dial 1996) from a custom button called *RunTDTSP* (see Figure 4.2). This COM object is an ATL project, and the resulting DLL is registered with ArcInfo. The custom button creates an instance of the object interface called *TDTSPLib.Tsp_alg*, which is used to make calls to the DLL's solution algorithm. The TDTSP algorithm stores its solution in a text file called TDTSPtour. The user can then display the tour results onscreen, and store the position number of each tour link in its appropriate *TourLinks#* feature class table. This is accomplished through the custom *DisplayTour* button control (see Figure 4.2).

When the user executes the *DisplayTour* button after running the *RunTDTSP,* the text file TDTSPtour is read in to populate the geodatabase. Which feature class, *TourLinks#,* the link belongs to is determined, and the feature that stores that link is located to fill the attribute fields 'Position' and 'Acc_Time' (see Figure 4.6). This link segment of the TDTSP tour solution is then displayed on the screen (see Figure 4.5), which is a cartographic display line and is only temporary. As each link is read

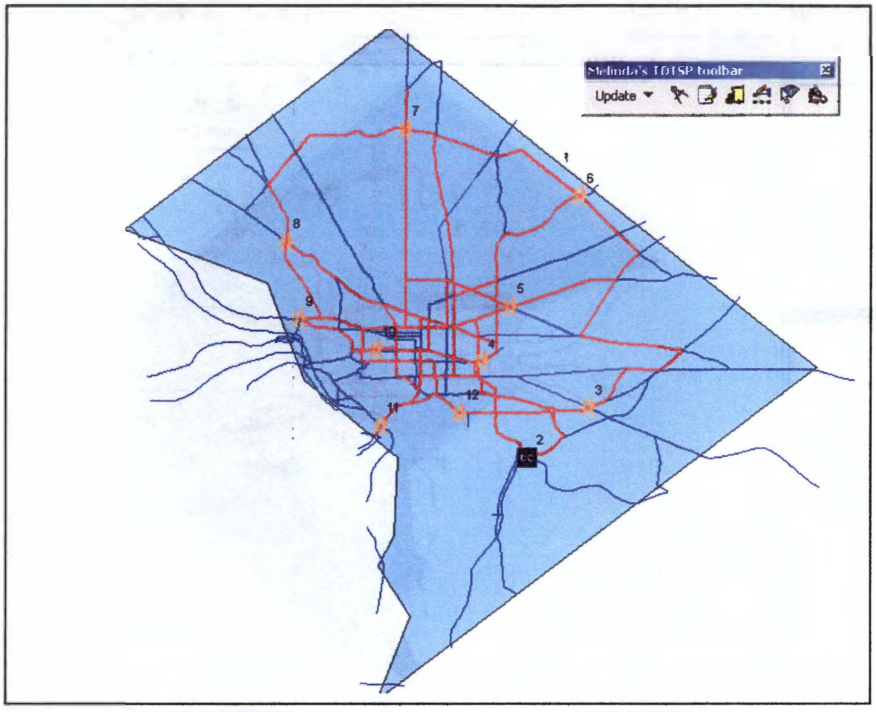


Figure 4.3 View of map after executing *BuildMatrices*.

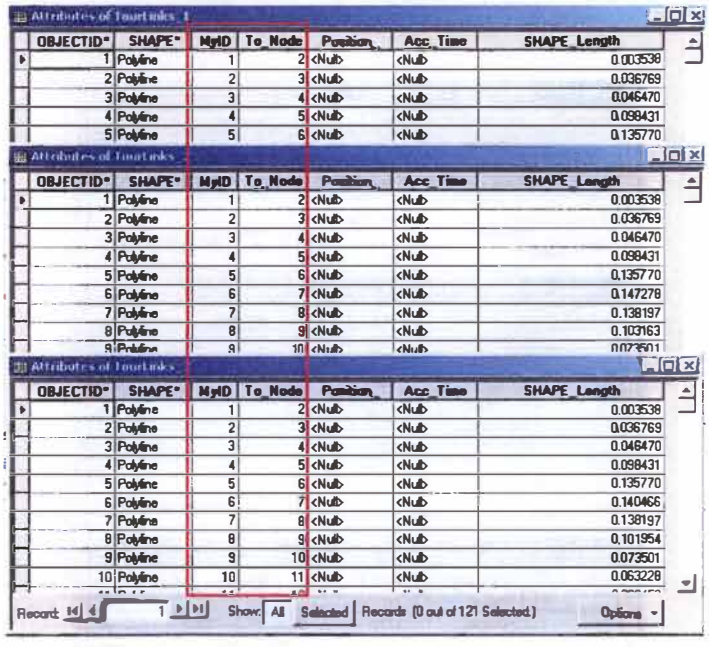


Figure 4.4 View of each *TourLinks#*'s table after executing *BuildMatrices*.

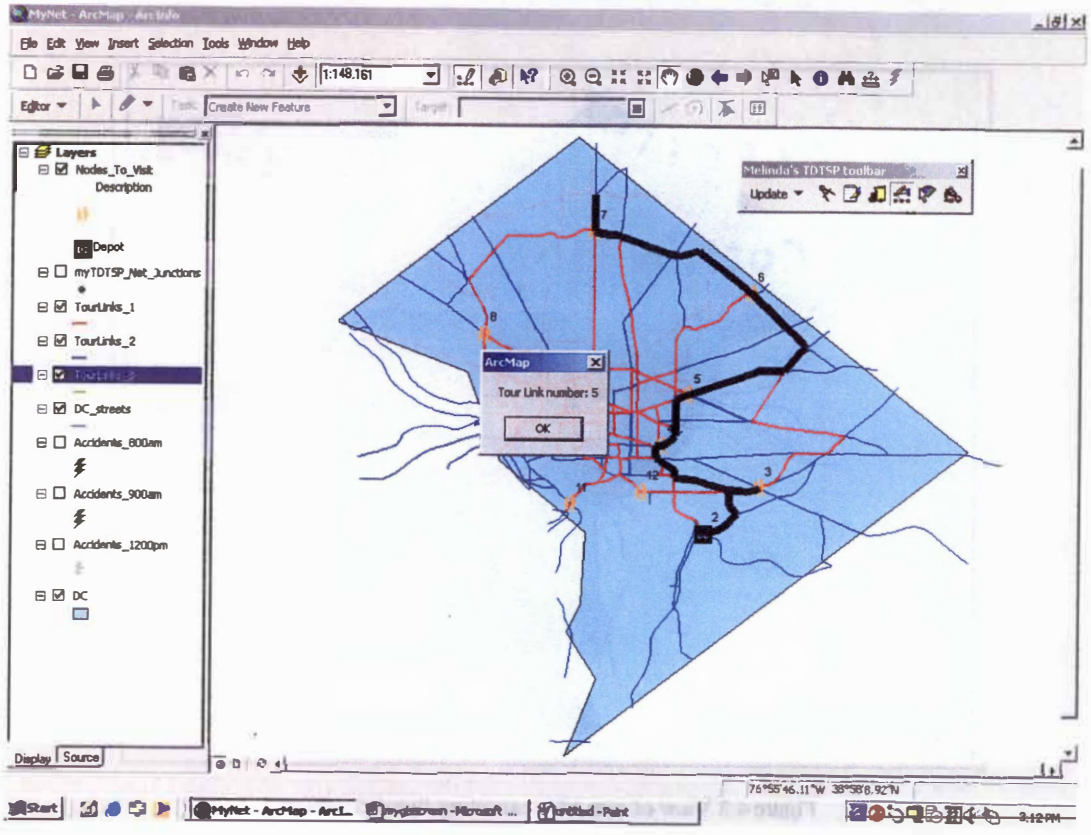


Figure 4.5 View of *DisplayTour* button in action.

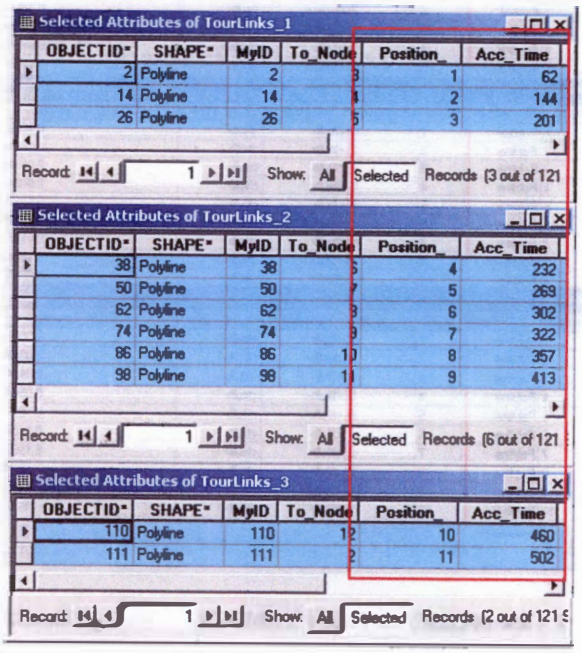


Figure 4.6 View of each *TourLinks#*'s table after executing *DisplayTour*.

in from the text file TDTSPtour, it is added to the visual display along with a message box stating the order this link is visited in the tour. *DisplayTour* ends by displaying a message box stating the overall cost of the TDTSP solution tour.

Using the information derived from the TDTSPtour text file the user can perform a *Select By Attribute* on each of the *TourLinks#* attribute field 'Position' to only display the links used by the tour (see Figure 4.6). To view each link used by the TDTSP tour solution as an individual symbol, the user can utilize ArcInfo's feature class drawing renderer to classify by unique field values (see Figure 4.1). Since only 10 nodes are to be serviced, there will be 11 unique line symbols.

This discussion has so far covered the procedures to solve the TDTSP problem. The next step is to handle any stochastic travel events that have the potential to affect the cost of the solution tour. This thesis study focuses chiefly on the stochastic travel events of vehicle accidents that directly affect the street network. As already mentioned in Chapters 2 and 3 there are numerous references to these stochastic events causing variations in travel speed in the literature. However, despite the possibility of such events they have not been widely incorporated into the TDTSP model. To prevent the TDTSP tour from deteriorating into a more costly tour, the nodes not yet reached on the tour can be Re-Ordered to avoid network disruptions and thereby provide a cheaper alternative. I assume that the traveling salesman can access the latest hourly traffic reports upon arriving at his most recent stop. The accidents are stored in a point feature class based on their locations and also contain their associated attributes of the delay they have caused in minutes. For my demonstration, the work period starts at 7 a.m. and ends at 6 p.m., so the first report is at 8 a.m. The

accidents are stored in the feature class called *Accidents_800am*. For demonstration purposes, I assumed at least three accidents per report, which are randomly located in the street network for this study. To determine if these accidents possibly affect the TDTSP tour the user makes use of the custom menu *Update* that contains the tools *Select By Attribute* and *Select By Location* (see Figure 4.1). The first step is to use the *Select By Attribute* tool to identify the "pseudo-networks" links that are used by the TDTSP solution tour. The second step uses the tool *Select By Location* to identify the "pseudo-network" links affected by accidents (see Figure 4.7). If there are accidents on links used by the tour, the next step is to locate the node the traveling salesman is at currently or traveling toward, which is determined for the user from the custom *MyUserUpdateForm* macro button. The *MyUserUpdateForm* macro button launches a custom VBA form to obtain the necessary information to run the *GetUpdates* procedure (see Figure 4.8). This routine determines if the links affected by accidents have not been reached yet by the tour. If they have not been reached yet and the links are within a two hour window, then the user will be prompted with a message box stating that the Re-Ordering algorithm should be run to determine if a better sequencing of nodes is available. The two hour time window is an assumption I have made for how long an accident can continue to affect the street network after occurring. After this prompt, the *MyUserUpdateForm* macro outputs information to a text file, called ReOrder, to be used by the Re-Ordering algorithm.

If the user receives the message prompt to execute the Re-Ordering algorithm after *MyUserUpdateForm,* the user then clicks on the *RunRe-Order* custom button (see Figure 4.2). By doing so, an instance of the custom created COM object

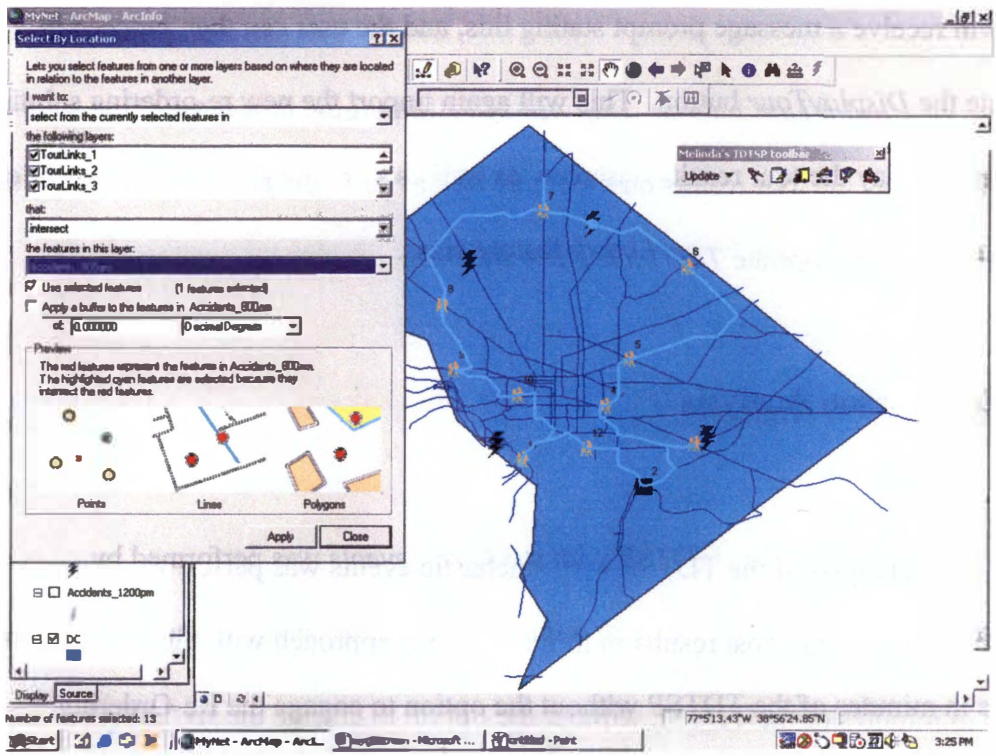


Figure 4.7 View of procedure used to select affected tour links.

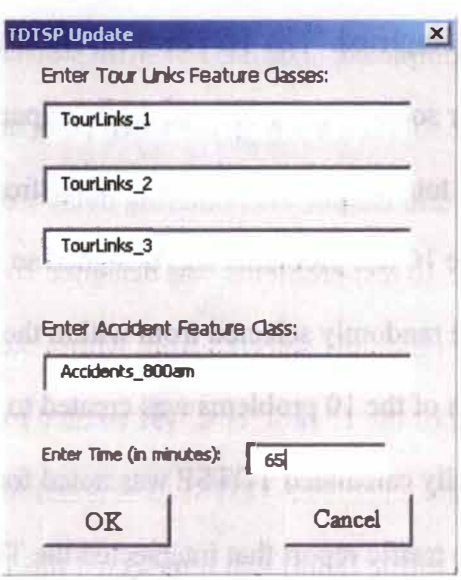


Figure 4.8 View of *MyUserUpdateForm*.

interface, called *TDTSPLib.ReOrder_alg*, is created that will call the methods for solving the Re-Ordering problem. When the Re-Ordering algorithm is completed, the user will receive a message prompt stating this, and the user can then proceed to execute the *DisplayTour* button. This will again import the new re-ordering solution tour and display the tour results onscreen, as well as store the position number of each tour link in its appropriate *TourLinks#* feature class.

4.2 Comparison Analysis

The analysis of the TDTSP with stochastic events was performed by comparing the overall cost results in minutes of this approach with the overall cost results in minutes of the TDTSP without the option to engage the Re-Ordering algorithm for 10 instances of the TSP problem. The overall cost for the TDTSP that adhered to the original tour also includes any delay values from accidents this tour encountered while being completed. The TDTSP with stochastic travel events successfully found cheaper solution alternatives when compared to the option of committing to the original tour despite encountering delay from accidents.

The structure of the 10 test problems was designed so that the original depot and 10 additional stops are randomly selected from within the *DCstreet* network. The "pseudo-network" for each of the 10 problems was created to solve the TDTSP. The overall cost for the originally calculated TDTSP was noted for further reference. If there were accidents in the traffic report that intersected the TDTSP solution tour and there was sufficient time to Re-Order (meaning the traveling salesman was not

already traveling on that link of the tour, had not already visited this link, and was within the two hour window of coming upon that link), these delay values of the accidents(s) were included in the overall cost of the option to not engage the Re-Ordering algorithm. The custom macro button called *MyUserUpdateForm* determines if the accidents fall within the two hour window constraint, if the traveling salesman is currently on that link of the tour, and if said salesman has not already visited these links prior to the previously mentioned traffic report. If I encounter a message box after executing the *MyUserUpdateForm* stating that Re-Ordering was necessary, I added to the overall cost of the option to not run the Re-Order algorithm the delay in minutes for the accident affecting the tour. For the Re-Ordering method, I executed the Re-Order algorithm and noted the new total overall travel cost. After receiving the last traffic accident report and determining if Re-Ordering was necessary, I took the final computed cost after the last Re-Order procedure was run as the final overall travel cost. The two travel cost were compared, and I found that the Re-Ordering method overall had the cheaper approach, except for one instance where the tour never encountered any accidents that meet the constraints for Re-Ordering. The results are displayed in table 4.1.

The results show that the Re-Ordering strategy did find a cheaper overall travel cost as compared to the option of not executing the Re-Order algorithm after finding an accident that affected the tour. Since the structure of my implementation allows the user to capture the number of links per the 3 time periods, I observed for the 10 test problems the distribution of these links among the time periods. The

Table 4.1 Comparison analysis results.

	Original Total	Re-Order Total	Non Re-Order Total	# of Accidents	Time Saved
Test1	502	539	552	2	13
Test2	530	550	560	1	10
Test3	498	498	498	0	0
Test4	478	511	526	2	15
Test5	526	565	579	2	14
Test6	568	581	598	1	17
Test7	458	462	478	1	16
Test8	534	573	584	2	11
Test9	482	524	538	2	14
Test10	521	544	550	1	6
					Average
					11.6

randomly generated stops to visit by the TDTSP varied spatially enough that during Test 4, there was only one link that fell during the last time period. The majority of the test cases had three links fall into time period 1, and during Test 5 there were four links that fell into this time period. In reality, the spatial dispersion of the stops to solve the TDTSP will vary, and there will be times when the links will not be evenly distributed among the time periods. There was one instance (Test 3) where the originally computed solution tour never came upon an accident that required Re-Ordering. This situation of course will happen in reality, which is why I first check to see if there is a pressing need to Re-Order if an accident(s) does fall on one of the network streets used by my TDTSP tour solution. The other 9 tests encountered between one and two accidents, and had accumulated delay values between 20 to 56 minutes. The average delay time for the 9 tests that encountered accidents averaged about 41 minutes, and the average time saved from using the Re-Order alternative solution was 11.4 minutes.

These are simplistic observation techniques, but I believe that this improvement in overall travel cost is enough reason to validate research into this component of variation in travel speeds. Also, I believe this is the first implementation strategy for including the stochastic travel events into the TDTSP model, and assumptions that were made for this study leave room for future research to continue to enhance the usefulness of the implementation of this component. Further computational analysis on stochastic travel events is also needed to capture a better understanding of the effectiveness of this component to the TSP problem.

Chapter 5

Conclusions and Future Research

The TSP has long spawned a great deal of research. This network problem has evolved from its first very generalized definition to the current research of considering time windows, dynamic custom ordering environments, stochastic travel speed variations, and now stochastic travel events affecting travel speeds. The evolution of the TSP problem reflects the goal of making it more reflective of real-world situations. The journey of the TSP through the literature has been a constant fine-tuning of the problem definition to include real world components that affect the problem formulation and solution. This thesis has effectively used the architectural framework of a GIS to develop a TDTSP module that also considers stochastic travel events. The motivation for moving the TSP into a GIS system is to utilize the GIS's DBMS for handling spatial and non-spatial data and analysis tools to use on the spatially referenced data inherent with this network problem. The majority of this success is due to ArcInfo 8's COM-based architecture that allowed me to create a seamless integration of these custom tools for solving this problem.

This study has shown that stochastic events also need to be considered in the journey toward developing a more realistic TSP model. The comparison analysis suggests including the stochastic component to be important for improving the applicability of the TSP to the real world. My strategy for handling the stochastic events is the beginning step towards a TSP model that can handle dynamic travel

speed variations effectively, with the intent of providing the users with more realistic solutions. Due to this being an initial step, there were some assumptions made to allow for a timely implementation. Future research is needed to continue the evolving process of developing a TSP solution that considers both the dependent and stochastic travel events.

For this study, I only considered a very specific stochastic travel event of traffic accidents that directly affected the street network. A future research direction needs to include all the other stochastic travel events into my GIS implementation of this TSP problem. Other stochastic events are indirect traffic accidents, such as vehicles on the side of the road, inclement weather conditions, and road construction.

Another consideration for adjustment is applicable to both the dependent and the stochastic travel event components. The assumption for this study, as well as for other studies (Hill and Benton 1992, Malandraki and Dial 1996, Picard and Queyranne 1978, and Fox, Gavish and Graves 1980), has been to index time periods by nodes. This approach has the inherent problem of not considering the possibility of the salesman crossing into another time period while traveling from origin node *i* to destination node *j*. The indexing by nodes approach uses the accumulated travel time at the origin node *i* to determine which time period cost matrix to use for this link *(i,j)*. In reality, travel speeds along a link are not constant; they can be affected by the crossing of time period boundaries that cause variations in speed along that link. Ichoua, Gendreau and Potvin (2002) developed an approach to handle the node indexing assumption setbacks. Their research focused on enforcing "first-in-first-out" (FIFO) property for the time-dependent models, which lead them to index travel

speeds by time periods and links. They used the approach of Taillard et al.'s (1997) parallel tabu search heuristic to handle the added computational complexity of indexing by links. This time period indexing strategy, coupled with handling stochastic travel events, is worth looking into for further research.

Another approach is to possibly modify Malandraki and Dial's (1996) restricted dynamic programming heuristic, which served as the basis to my thesis, to handle indexing time periods by links instead of by nodes. An additional reason to support this research direction is that Malandraki and Dial found that this heuristic outperformed previously known heuristics in solution quality, execution time, and size of the problem solved. Another consideration for my TDTSP heuristic algorithm is the possibility of storing a larger number of subtours, H, at each iteration. Malandraki and Dial tested the restricted algorithm for user-specified subtour size of 1, 100, 1000, 5000 and 15000, with nodes ranging from 10 up to 55. As expected, the algorithm solution improved, in respect to reaching the optimal solution, with the increase in the number of partial tours saved. For my implementation, I only stored up to 100 partial tours, which were found to have a 13.8% average improvement as compared to only saving 1 partial tour. If the number of partial tours saved was increased to 15000, Malandraki and Dial found the improvement of the know solution for the approach to save only 100 subtours increased by 4.8%. This increase in the number of subtours saved could provide users with a better TDTSP solution. This future research direction is a viable consideration while searching for an algorithm to provide a better quality solution and strategy in indexing by time periods.

The approach I used to handle stochastic events can be improved by not only considering Re-Ordering of the node visitation sequence, but to also consider whether there exists an alternative path for traveling from some origin node *i* to some destination node *j* to avoid traffic accidents. This would provide even more alternatives in preventing the solution tour from increasing in travel cost. This approach would locate the links affected by accidents that the traveling salesman has not yet reached and dynamically calculate a new shortest path based on the current traffic speed conditions. The possibility of the original path remaining the cheapest does exist but this would probably only occur in sparse street networks. After the dynamic shortest path has been calculated, the Re-Ordering heuristic can still be invoked to see if, even with this new path for link *(i,j)*, a new node sequence might provide an even cheaper alternative to the current situation. This can be confirmed by performing additional computational analysis.

The graphic user interface can be further developed. Determining beneficial and efficient graphics is the research focus of the field of Image Theory (IT) (Crossland et al. 1995). Bodin and Levy (1994) incorporate the research of IT when they list high quality graphics as part of their requirements for developing a computer-assisted Vehicle Routing Schedule (VRS) problem in a GIS environment. In my study the GIS environment completely handles the visualization of my TDTSP implementation with stochastic events. The GIS also provides the means to manage the spatially referenced data used to solve the TDTSP. The specific part of this graphic user design that can be enhanced is the actual user interface procedures I developed in ArcInfo 8. For this study, the feature classes (*TourLinks#*) used to store

the TDTSP's "pseudo-network" and their associated attributes are hard coded into the VBA routines *BuildMatrices* and *RunTDTSP*. This implementation would be much more useful and efficient if it was designed similar to the *MyUserUpdateForm*, which when executed, launches an input form to obtain the names of the feature classes to use in the VBA routine. This is obviously more advantageous then having the user hard code these names in, or to always use the same named feature classes. An improved scenario would be for an input form to be launched when *BuildMatrices* is selected. This form is used to receive information on:

- the names of the feature classes in which the shortest path calculations will be saved;
- the feature class with the stops for which the TDTSP must be solved; and
- the feature class that holds the network weights and their field names.

Additionally, it would be simpler for the user not to have to create the feature classes that store the shortest path calculations in ArcCatalog, but to instead rely on the code to create and add these feature classes to the map document. The same type of input form for the *RunTDTSP* would also improve and simplify the user interface.

A last future direction for my implementation strategy is to actually obtain the traffic reports through a traffic update website, such as an ITS. ITS websites are becoming commonplace and increasing in the detail that they can provide to users, and would be a direction to pursue implementing a download stream to my ArcInfo project. This would further show the benefits of including stochastic events when solving the TDTSP, as well as continue the TSP's applicability to the real world.

REFERENCES

References

Ahuja, Ravindra K., Thomas L. Magnanti and James B. Orlin. 1993. *Network Flows: Theory, Algorithms and Applications.* Prentice Hall, New Jersey.

Barber, Gerald. 1995. Aggregate Characteristics of Urban Travel. In Hanson, Susan (Ed.), *The Geography of Urban Transportation*: 81-99. The Guilford Press, New York.

Barbosa, L. C. and R. G. Hirko. 1980. Integration of algorithmic aids into a decision support systems. *MIS Quarterly* 4(1): 1-12.

Bennett, David A. 1997. A framework for the integration of geographical information systems and modelbase management. *International Journal of Geographical Information Science* 11(4): 337-357.

Bertsimas, Dimitris and et al. 1995. Computational approaches to stochastic vehicle routing problems. *Transportation Science* 29(2): 342-352.

Bianco, Lucio, Aristide Mingozzi and Salvatore Ricciardelli. 1993. The traveling salesman problem with cumulative cost. *Networks* 23: 81-91.

Bodin, Lawrence and Laurence Levy. 1994. Visualization in vehicle routing and scheduling problems. *ORSA Journal on Computing* 6(3): 261-268.

Bowman, E. H. 1956. Production scheduling by the transportation method of linear programming. *Operations Research* 4: 100-103.

Castillo, Enrique, Jose Manual Gutierrez and Ali S. Hadi. 1997. *Expert Systems and Probabilistic Network Models*. Springer, NewYork.

Crossland, M. D., B. E. Wynne and W. C. Perkins. 1995. Spatial decision support systems: an overview of technology and a test of efficacy. *Decision Support Systems* 14: 219-235.

Fox, Kenneth R., Bezalel Gavish and Stephen C. Graves. 1980. An *n*-constraint formulation of the (time-dependent) traveling salesman problem. *Operations Research* Vol. 28, No. 4: 1018-1021.

Fu, Liping. 2001. An adaptive routing algorithm for in-vehicle route guidance systems with real-time information. *Transportation Research B* 35: 749-765.

Gendreau, Michael, Gilbert Larporte and Rene Seguin. 1995. An exact algorithm for the vehicle routing problem with stochastic demands and customers. *Transportation Science* 29(2): 143-155.

Golden, B. L. and A. A. Assad (eds.). 1988. *Vehicle Routing: Methods and Studies*. North-Holland, Amsterdam.

Goodchild, Michael, Robert Haining and Stephen Wise. 1992. Integrating GIS and spatial data analysis: problems and possibilities. *International Journal of Geographical Information Systems* 6(5): 407-423.

Hafer, Lou. 2001. bonsaiG. http://www.cs.sfu.ca/~lou/BonsaiG/index.html.

Hickman, Mark D. and David H. Bernstein. 1997. Transit service and path choice models in stochastic and time-dependent networks. *Transportation Science* 31(2): 129-146.

Hill, Arthur V., and W. C. Benton. 1992. Modeling intra-city time-dependent travel speeds for vehicle scheduling problems. *Journal of Operational Research Society* 43(4): 343-351.

Hu, Patricia S. and Jennifer R. Young. 1999. *Summary of Travel Trends 1995 NPTS.* http://www-cta.ornl.gov/npts/1995/Doc/publications.shtml.

Ichoua, Soumia, Michael Gendreau and Jean-Yves Potvin. 2002. Vehicle dispatching with time-dependent travel times. *European Journal of Operational Research* Article In Press.

Keenan, Peter B. 1998. Spatial decision support systems for vehicle routing. *Decision Support Systems* 22: 65-71.

Jones, Christopher V. 1994. Visualization and optimization. *ORSA Journal on Computing* 6(3): 221-257.

Lawler, E. L., J. K. Lenstra, Kan Rinnooy, A. H. G. and D. B. Shmoys. (eds.) 1985. *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization.* John Wiley & Sons, New York.

Lewis, Simon. 1990. Use of Geographical Information Systems in Transportation Modeling. *ITE Journal* 60(March): 34-38.

Lucena, Abilio. 1990. Time-dependent traveling salesman problem- the deliveryman case. *Networks* 20: 753-763.

Malandraki, Chryssi and Robert B. Dial. 1996. A restricted dynamic programming heuristic algorithm for the time dependent traveling salesman problem. *European Journal of Operational Research* 90: 45-55.

Makhorin , Andrew. 2001. GLPK. http://www.gnu.org/software/glpk/glpk.html. Department for Applied Informatics, Moscow Aviation Institute, Moscow, Russia.

Malandraki, Chryssi and Mark S. Daskin. 1992. Time dependent vehicle routing problems: Formulations, properties and heuristic algorithms. *Transportation Science* 26(3): 185-200.

Miller, Harvey J., Yi-Hwa Wu and Ming-chih Hung. 1999. GIS-based dynamic traffic congestion modeling to support time-critical logistics. In: *IEEE Proceedings of the Hawaii international Conference on System Science*, Maui Hawaii.

Miller, H. and Shaw, S.L. 2001. *Geographic Information Systems for Transportation: Principles and Applications*. New York: Oxford University Press.

Moscato, Pablo. 2000. TSPBIB. http://www.densis.fee.unicamp.br/~moscato/TSPBIB_home.html. Universidade Estadual de Campinas.

Nyerges, Timothy L. 1992. Coupling GIS and spatial analytical models. *Proceedings of the 5th International Symposium on Spatial Data Handling* 2: 534-543.

Park, Yang-Byung and Sung-Hun Song. 1997. Vehicle scheduling problems with time-varying speed. *Computers Ind. Engng* 33(3-4): 853-856.

Peetra, Srinivas and Hani S. Mahmassani. 1995. Multiple user classes real-time traffic assignment for online operations: a rolling horizon solution framework. *Transportation Research C* 3(2): 83-98.

Picard, Jean-Claude and Maurice Queyranne. 1978. The time-dependent traveling salesman problem and its application to the tardiness problem in one-machine scheduling. *Operations Research* 26(1): 86-110.

Ralston, B.A. and J. J. Ray. 1987. FIELDS: The food for peace inland emergency logistics and distribution center. Developed for the United States Agency for International Development.

Ralston, B.A., C. Liu, and M. Zhang. 1993. TRAILMAN: The transportation and inland logistics manager. Developed for the World Food Programme.

Ralston, B. A., G. Tharakan, and C. Liu. 1994. A spatial decision support system for transportation policy analysis, *Journal of Transport Geography* 2: 101-110.

Reinelt , Gerhard. 2001. TSPLIB. http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/. Universität Heidelberg.

Slater, Alan. 2002. Specifications for a dynamic vehicle routing and scheduling system. *International Journal of Transportation Management* 1: 29-40.

Solomon, Marius M. 1987. Algorithms for vehicle routing and scheduling problems with time window constraints. *Operations Research* 35(2): 254-265.

Sutton, John. 1997. Data attributions and network representation issues in GIS and Transportation. *Transportation Planning and Technology* 21: 25-44.

Taillard, E., P. Badeau, M. Gendreau, F. Guertin and Y. L. Potvin. 1997. A tabu search heuristics for vehicle routing problem with soft time windows. *Transportation Science* 31: 170-186.

Tarantilis, C. D. and C. T. Kiranoudis. 2002. Using a spatial decision support system for solving the vehicle routing problem. *Information and Management* 39: 359-375.

Ungerer, Matthew J. and Michael F. Goodchild. 2002. Integrating spatial data analysis and GIS: a new implementation using Component Object Model (COM). *International Journal of Geographical Information Science* 16(1): 41-53.

Wu, Yi-Hwa, Harvey J. Miller and Ming-chih Hung. 2001. A GIS-based decision support system for analysis of route choice in congested urban road networks. *Journal of Geographical Systems* 3: 3-24.

Zeiler, Michael (ed.). 2001. *Exploring ArcObjects.* Environmental Systems Research Institute, Inc, Redlands, California.

APPENDIX

List of Appendices

Appendix 1 TDTSP heuristic algorithm code 102

Appendix 2 Re-Order heuristic algorithm code 110

Appendix 3 ArcInfo 8 VBA macro code 120

Appendix 1

TDTSP heuristic algorithm code

```
// TSP_alg.cpp : Implementation of CTSP_alg
#include "stdafx.h"
#include "TDTSP.h"
#include "TSP_alg.h"
#include "Tour.h"

#include <iostream.h>
#include <iomanip.h>
#include <fstream.h>

////////////////////////////////////////////////////////////////////////////
// CTSP_alg


int CTSP_alg::getNewNode(int tsMatrix[][11], int marked[], int lastnode, int &cost)
{
        //function to select an unmarked node not in subtour yet
        int node;
        for(int i = 0; i < 11; i++)
        {
                if(tsMatrix[i][lastnode] != 0 && marked[i] == 0)
                        {
                                node = i;
                                cost = tsMatrix[i][lastnode];
                                break;
                        }
        }
        return node;

}

void CTSP_alg::FindTour(int TourResult[])
{
        //class function to solve the TDTSP
        //the time starts at 0 minutes (7:00am)
        //the breaks in time are hard coded here as local variables
        const int time1 = 210;
        const int time2 = 420;
        const int time3 = 600;

        int i, j;
```

```
//Initializing Phase
Tour subtour[10]; //this used for tour length of 2
int node, cost, lastnode;
Tour *sub[10];
for(i = 0; i < 10; i++)//initialize pointers
{
        sub[i] = 0;
}


ofstream fout;
//open file to store the final results
fout.open("c:\\temp\\tdtsptour.txt" , ios::out);

int marked[11];

marked[0] = 1;
for(i = 1; i < 11; i++)//initailize marked array
{
        marked[i] = 0;
}

node = 0;//array start counts at zero
cost = 0;
        int secondnode = 1;
for(i = 0; i < 10; i++)
{
        subtour[i].buildtour(sub[i], node, cost);
        subtour[i].initializeTour(sub[i],tsMatrix1, secondnode);
        secondnode++;
}

int tourlength = 2;
//create potential new tours to keep
int num = 11;
int worsttotal = 0;
int worstindex;
Tour potentials[100];
Tour *psub[100];
Tour prevtour[100];
Tour *prev[100];

for(i = 0; i < 100; i++)
{
        psub[i] = 0;
        prev[i] = 0;
}

node = 0;//array start counts at zero
```

```
cost = 0;

for(i = 0; i < 100; i++)
{
        potentials[i].buildtour(psub[i], node, cost);
        prevtour[i].buildtour(prev[i], node, cost);

}
int pcount = 0;

int temptotal;
Tour *temp = 0;
for(i = 0; i < 10; i++)
{
        lastnode = subtour[i].lastNode(sub[i]);
        temp = sub[i];
        for(int ii = 1; ii < 11; ii++)
        {
                marked[ii] = 0;
        }
        for(j = 0; j < (num - tourlength); j++)
        {
                if(pcount < 100)
                {
                        marked[lastnode] = 1;
                        potentials[pcount].copyTour(psub[pcount], temp);
                        //copy initial subtours to potentials array
                        node = getNewNode(tsMatrix1, marked, lastnode, cost);
                        marked[node] = 1;
                        potentials[pcount].buildtour(psub[pcount], node, cost);
                        temptotal = potentials[pcount].getTotal(psub[pcount]);
                        if(worsttotal < temptotal)//set worst subtour variables
                        {
                                worsttotal = temptotal;
                                worstindex = pcount;
                        }
                        pcount++;
                }//end if
                else
                {
                        marked[lastnode] = 1;
                        node = getNewNode(tsMatrix1, marked, lastnode, cost);
                        temptotal = subtour[i].getTotal(sub[i]);
                        //see if need to replace worstindex subtour
                        if(worsttotal > (temptotal + cost))
                        {
                potentials[worstindex].resetTour(psub[worstindex], sub[i], node,
cost);
                                worsttotal = 0;
```

```
                                        for(int k = 0; k < 90; k++)
                                        {       temptotal = potentials[k].getTotal(psub[k]);
                                                if(temptotal > worsttotal)
                                                {
                                                        worsttotal = temptotal;
                                                        worstindex = k;
                                                }
                                        }
                                }//end if
                                marked[node] = 1;
                        }//end else
                }//end for j
        }//end for i

        //SubTour Building Phase
        tourlength++;
        pcount = 0;
int tcount = 90;
        for(i = 0; i < 90; i++)
        {
                prevtour[i].copyTour(prev[i], psub[i]);//initialize prevtour for this iteration
        }
        for(int k = 0; k < 8; k++)
        {

temp = 0;
        for(i = 0; i < tcount; i++)
        { //fout << " I "<< i << "    ";
                lastnode = prevtour[i].lastNode(prev[i]);
                for(int ii = 1; ii < 11; ii++) //re-initialize marked array
                {
                        marked[ii] = 0;
                }
                int tempnum;
                temp = prev[i];
                for(int kk = 0; kk < tourlength; kk++)//mark all the nodes already in subtour
                {
                        tempnum = prevtour[i].NextNode(temp);
                        marked[tempnum] = 1;
                }
                for(j = 0; j < (num - tourlength); j++)
                {
                        if(pcount < 100)
      //prevtour is full and now must see if new subtour sequence better than worstindex
                        {
                                if(k == 0 && pcount > 89)
                                {
                                        potentials[pcount].copyTour(psub[pcount], prev[i]);
                                        prevtour[pcount].copyTour(prev[pcount], prev[i]);
```

```
                    }
                    else
                            potentials[pcount].resetTour2(psub[pcount],
prev[i]);
                    temptotal = potentials[pcount].getTotal(psub[pcount]);
                    if(temptotal < time1)
                            node = getNewNode(tsMatrix1, marked, lastnode,
cost);
                    else
                    {
                            if(temptotal < time2)
                            node = getNewNode(tsMatrix2, marked, lastnode,
cost);
                            else
                            {
                                    //it is in time3
                            node = getNewNode(tsMatrix3, marked, lastnode,
cost);
                            }
                    }
                    marked[node] = 1;
                    potentials[pcount].buildtour(psub[pcount], node, cost);
                    temptotal = potentials[pcount].getTotal(psub[pcount]);
                    if(worsttotal < temptotal) / /set worst
                    {
                            worsttotal = temptotal;
                            worstindex = pcount;
                    }
                    pcount++;
            }//end if
            else
            {
                    temptotal = prevtour[i].getTotal(prev[i]);
                    marked[lastnode] = 1;
                    //get right time period travel speed cost
                    if(temptotal < time1)
                            node = getNewNode(tsMatrix1, marked, lastnode,
cost);
                    else
                    {
                            if(temptotal < time2)
                            node = getNewNode(tsMatrix2, marked, lastnode,
cost);
                            else
                            {
                                    //it is in time3
                            node = getNewNode(tsMatrix3, marked, lastnode,
cost);
                            }
```

```
                                }
                                if(worsttotal > (temptotal + cost))//found a cheaper tour
                                {
                        potentials[worstindex].resetTour(psub[worstindex], prev[i], node,
cost);
                                        worsttotal = 0;
                                        for(int k = 0; k < 100; k++)
                                        {       temptotal = potentials[k].getTotal(psub[k]);
                                                if(temptotal > worsttotal)
                                                { //reset worst index and cost
                                                        worsttotal = temptotal;
                                                        worstindex = k;
                                                }
                                        }//end for

                                }//end if
                                marked[node] = 1;
                        }//end else

                }//edn for j
        }//end for i
        tcount = 100;
        for(i = 0; i < 100; i++)
        {       //reset the new subtours in prevtour after this iteration
                lastnode = potentials[i].lastNode(psub[i]);
                cost = potentials[i].lastCost(psub[i]);
                prevtour[i].resetTour2(prev[i], psub[i]);
                prevtour[i].buildtour(prev[i], lastnode, cost);
        }
        tourlength++;
        pcount = 0;

        }//end for k

        //go back to the depot - node 0
        node = 0;
        int total;
        for(i = 0; i < 100; i++)
        {
                lastnode = prevtour[i].lastNode(prev[i]);
                cost = tsMatrix3[node][lastnode];
                prevtour[i].buildtour(prev[i], node, cost);
        }
        cost = 9999;
        int index;
        for(i = 0; i < 100; i++)
        {
                total = prevtour[i].getTotal(prev[i]);
                if(cost > total)
```

```
                {
                        cost = total;
                        index = i;
                }
        }
        prevtour[index].TopUp(prev[index], fout);
        fout << endl << cost << endl;
        return;
}

STDMETHODIMP CTSP_alg::TourSolution()
{
        //interface method
        int TourResults[12];
        FindTour(TourResults);

        return S_OK;
}

STDMETHODIMP CTSP_alg::import_Matrix()
{
//interface method to read in the 3 text files (path: c:\temp\) that contain the travel speed cost
//for each node to every other node
//for each of the three time periods and store in the tsMatrices arrays
        int i;
        int number;
        ifstream fin;
        fin.open("C:\\temp\\tsMatrix1.txt", ios::in);
        for(i = 0; i < 11; i++)
        {
                for(int t = 0; t < 11; t++)
                {
                        fin >> number;
                        tsMatrix1[i][t] = number;
                }

        }
        fin.close();

        fin.open("C:\\temp\\tsMatrix2.txt", ios::in);
        for(i = 0; i < 11; i++)
        {
                for(int t = 0; t < 11; t++)
                {
                        fin >> number;
                        tsMatrix2[i][t] = number;
                }

        }
```

```
        fin.close();

        fin.open("C:\\temp\\tsMatrix3.txt", ios::in);
        for(i = 0; i < 11; i++)
        {
                for(int t = 0; t < 11; t++)
                {
                        fin >> number;
                        tsMatrix3[i][t] = number;
                }
        }
        fin. close();
        return S_OK;
}
```

Appendix 2

Re-Order heuristic algorithm code

```
// Re_Order.cpp : Implementation of CRe_Order
#include "stdafx.h"
#include "TDTSP.h"
#include "Re_Order.h"
#include "Tour.h"

#include <iostream.h>
#include <iomanip.h>
#include <fstream.h>

//////////////////////////////////////////////////////////////////////////
// CRe_Order

int CRe_Order::get_ReNewNode(int tsMatrix[][11], int marked[], int lastnode, int &cost)
{
	//function to select an unmarked node not in subtour yet
	int node;
	for(int i = 0; i < 11; i++)
	{
		if(tsMatrix[i][lastnode] != 0 && marked[i] == 0)
			{
				node = i;
				cost = tsMatrix[i][lastnode];
				break;
			}
	}
	return node;
}

void CRe_Order::Find_ReTour(int TourResult[])
{
	//class function to solve the Re-Order
	//the time starts at 0 minutes (7:00am)
	//the breaks in time are hard coded here as local variables
	const int time1 = 210;
	const int time2 = 420;
	const int time3 = 600;

	int i;
	int marked[11];
	int staticM[11];
	marked[0] = 1; staticM[0] = 1;
	for(i = 1; i < 11; i++)
```

```
{
        marked[i] = 0;
        staticM[i] = 0;
}

ifstream fin;
Tour subtour[10];
int node, cost, lastnode, total;
int tempnum;
Tour *sub[10];
for(i = 0; i < 10; i++)
{
        sub[i] = 0;
}
//struct to use to store all the information for the links affected by accidents
struct LinksToReOpt {
        int linkID;
        int ToNode;
        int delayvalue;
};

LinksToReOpt *Links;
int LinksDone, time;
int tourlength;
int countLinks;
total = 0;

Tour BaseTour; //the last TDTSP solution tour and read in to use as the base tour
Tour *base; Tour *tempbase;
tempbase = 0;
base = 0;  int count = 0;

//Bring in Links with delay values
fin.open("C:\\temp\\ReOpt.txt", ios::in);
fin >> countLinks >> time;
Links = new LinksToReOpt[countLinks];

for(i = 0; i < countLinks; i++)
{
        fin >> Links[i].linkID >> Links[i].ToNode >> Links[i].delayvalue;
}
fin.close();

//Bring in original tour
fin.open("C:\\temp\\tdtsptour.txt" , ios::in);

for(i = 0; i < 12; i++)
{
        fin >> node >> cost;
```

```
                fout2 << "read in: " << node << cost;
                BaseTour.buildtour(base, node, cost);
                total += cost;
                if(total < time)//LinksDone can't be greater than 8
                {
                        LinksDone = count;
                 }

                for(int j = 0; j < countLinks; j++)
                {
                        if(count == Links[j].linkID)
                        {
                                        if(time < time1)
                                        {
        RetsMatrix1[lastnode][node] = Links[j].delayvalue + RetsMatrix1[lastnode][node];
                                        RetsMatrix1[node][lastnode] +=
Links[j].delayvalue;
                                        }
                                        else
                                                if(time > time2)
                                                {
                                        RetsMatrix3[lastnode][node] +=
Links[j].delayvalue;
                                        RetsMatrix3[node][lastnode] +=
Links[j].delayvalue;
                                                }
                                                else
                                                {       //its in time period 2
                                        RetsMatrix3[lastnode][node] +=
Links[j].delayvalue;
                                        RetsMatrix3[node][lastnode] +=
Links[j].delayvalue;
                                                }
                        }
                }
                lastnode = node;
                count++;
        }
        fin.close();
        tempbase = base;
        for(i = 1; i < 12; i++)
        {
                tempnum = BaseTour.Re_NextNode(base,cost);
                if(i <= LinksDone)
                {
                                fout2 << "tempnum " << tempnum << "i" << i << endl;
                                marked[tempnum] = 1;
                                staticM[tempnum] = 1;
                                total += cost;
```

```
        }
}
base = tempbase;

tourlength = LinksDone + 1;
ofstream fout;
fout.open("c:\\temp\\tdtsptour.txt" , ios::out); //open file to save the solution tour

node = 0;//array start counts at zero
cost = 0;
int nextnode;
for(i = 0; i < (10 - LinksDone); i++)
{
        subtour[i].buildtour(sub[i], node, cost);
        subtour[i].copyBase(sub[i], base, LinksDone, fout2);
        lastnode = subtour[i].lastNode(sub[i]);
        if(total < time1)
                nextnode = get_ReNewNode(RetsMatrix1, marked, lastnode, cost);
        else
        {
                if(total > time2)
                nextnode = get_ReNewNode(RetsMatrix3, marked, lastnode, cost);

                else
                nextnode = get_ReNewNode(RetsMatrix2, marked, lastnode, cost);
        }
        marked[nextnode] = 1;
        subtour[i].buildtour(sub[i], nextnode, cost);
        subtour[i].TopUp(sub[i], fout2);
}

tourlength++; fout2 << "tourlength" << tourlength; fout2.close();
//create potential new tours to keep
int num = 11;
int worsttotal = 0;
int worstindex;
Tour potentials[100];
Tour *psub[100];
Tour prevtour[100];
Tour *prev[100];

for(i = 0; i < 100; i++)
{
        psub[i] = 0;
        prev[i] = 0;
}

node = 0;//array start counts at zero
cost = 0;
```

```
for(i = 0; i < 100; i++)
{
        potentials[i].buildtour(psub[i], node, cost);
        prevtour[i].buildtour(prev[i], node, cost);
}
int pcount = 0;
int temptotal, j;
Tour *temp = 0;

//1st Step
for(i = 0; i < (10 - LinksDone); i++)
{
        lastnode = subtour[i].lastNode(sub[i]);
        temp = sub[i];
        for(int ii = 1; ii < 11; ii++)
        {
                marked[ii] = staticM[ii];
        }
        for(j = 0; j < (num - tourlength); j++)
        {
                marked[lastnode] = 1;
        potentials[pcount].copyTour(psub[pcount], temp);
                node = get_ReNewNode(RetsMatrix1, marked, lastnode, cost);
                marked[node] = 1;
                potentials[pcount].buildtour(psub[pcount], node, cost);
                temptotal = potentials[pcount].getTotal(psub[pcount]);
                if(worsttotal < temptotal)
                {
                        worsttotal = temptotal;
                        worstindex = pcount;
                }
                pcount++;
        }//end for j
}//end for i

tourlength++;
int tcount;
tcount = pcount;
int iteration;

//2nd Step
for(i = 0; i < tcount; i++)
{
        prevtour[i].copyTour(prev[i], psub[i]);
}
pcount = 0;
int ToGo = 11 - tourlength;
for(int k = 0; k < ToGo; k++)
```

```
{
  temp = 0;
        iteration = 1;
        for(i = 0; i < tcount; i++)
        {
                lastnode = prevtour[i].lastNode(prev[i]);
                for(int ii = 1; ii < 11; ii++)
                {
                        marked[ii] = 0;
                }
                temp = prev[i];
                int thiscost;
                for(int kk = 0; kk < tourlength; kk++)
                {
                        tempnum = prevtour[i].Re_NextNode(temp, thiscost);
                        marked[tempnum] = 1;
                }
                for(j = 0; j < (num - tourlength); j++)
                {
                if(pcount < 100)
                {
                        if(k == 0 && pcount >= tcount)
                        {
                                potentials[pcount].copyTour(psub[pcount], prev[i]);
                                prevtour[pcount].copyTour(prev[pcount], prev[i]);
                        }
                        else
                                potentials[pcount].resetTour2(psub[pcount],
prev[i]);
                        temptotal = potentials[pcount].getTotal(psub[pcount]);

                        if(temptotal < time1)
                        node = get_ReNewNode(RetsMatrix1, marked, lastnode,
cost);
                        else
                        {
                                if(temptotal > time2)
                        node = get_ReNewNode(RetsMatrix3, marked, lastnode,
cost);
                                else
                                {
                                        //it is in time2
                        node = get_ReNewNode(RetsMatrix2, marked, lastnode,
cost);                                  }
                }
                        marked[node] = 1;
                        potentials[pcount].buildtour(psub[pcount], node, cost);

                        temptotal = potentials[pcount].getTotal(psub[pcount]);
```

```
                        if(worsttotal < temptotal)
                        {
                                worsttotal = temptotal;
                                worstindex = pcount;
                        }
                        pcount++;
                }//end if pcount < 100
                else
                {
                        temptotal = prevtour[i].getTotal(prev[i]);
                        marked[lastnode] = 1;

                        //get right time period travel speed cost
                        if(temptotal < time1)
                        node = get_ReNewNode(RetsMatrix1, marked, lastnode,
cost);
                        else
                        {
                                if(temptotal < time2)
                        node = get_ReNewNode(RetsMatrix2, marked, lastnode,
cost);
                                else
                                {
                                        //it is in time3
                        node = get_ReNewNode(RetsMatrix3, marked, lastnode,
cost);
                                }
                        }
                        if(worsttotal > (temptotal + cost))//found a chaper tour
                        {
                potentials[worstindex].resetTour(psub[worstindex], prev[i], node,
cost);
                                worsttotal = 0;
                                for(int k = 0; k < 100; k++)
                                {       temptotal = potentials[k].getTotal(psub[k]);
                                        if(temptotal > worsttotal)
                                        { //reset worst index and cost
                                                worsttotal = temptotal;
                                                worstindex = k;
                                        }
                                }//end for
                        }//end if
                        marked[node] = 1;
                }//end else
        }//end for j
}//end for i

if(LinksDone < 5)//reset how many iterations have to be done
        //depending on how many nodes already visited by the tour at this moment
```

```
            tcount = 100;
    else
    {
            if(iteration = 1)
            {
                    if(LinksDone = 5)
                            tcount = 80;
                    if(LinksDone = 6)
                            tcount = 36;
                    if(LinksDone = 7)
                            tcount = 12;
            }
            else
            {
                    if(LinksDone = 5)
                            tcount = 100;
                    if(LinksDone = 6)
                            tcount = 72;
            }
    }
    for(i = 0; i < tcount; i++)
    {  //set new subtours in prevtour
            lastnode = potentials[i].lastNode(psub[i]);
            cost = potentials[i].lastCost(psub[i]);
            prevtour[i].resetTour2(prev[i], psub[i]);
            prevtour[i].buildtour(prev[i], lastnode, cost);

    }
    tourlength++;
    pcount = 0;
    iteration++;

    }//end for k to ToGo

    //go back to the depot - node 0
    node = 0;

    for(i = 0; i < 100; i++)
    {
            lastnode = prevtour[i].lastNode(prev[i]);
            cost = RetsMatrix3[node][lastnode];
            prevtour[i].buildtour(prev[i], node, cost);
    }

    cost = 9999;
    int index;
    for(i = 0; i < 100; i++)
    {
            total = prevtour[i].getTotal(prev[i]);
```

```
                if(cost > total)
                {
                        cost = total;
                        index = i;
                }
        }
        prevtour[index].TopUp(prev[index], fout);
        return;
}

STDMETHODIMP CRe_Order::import_Re_Matrix()
{
//interface method to read in the 3 text files (path c:\temp\) that contain the travel speed cost
//for each node to every other node
//for each of the three time periods and store in the tsMatrices arrays
        int i;    ofstream fout2;
        int number;
        ifstream fin, fin2;
        fin.open("C:\\temp\\tsMatrix1.txt", ios::in);
        for(i = 0; i < 11; i++)
        {
                for(int t = 0; t < 11; t++)
                {
                        fin >> number;
                        RetsMatrix1[i][t] = number;
                }
        }
        fin.close();

        fin.open("C:\\temp\\tsMatrix2.txt", ios::in);
        for(i = 0; i < 11; i++)
        {
                for(int t = 0; t < 11; t++)
                {
                        fin >> number;
                        RetsMatrix2[i][t] = number;
                }
        }
        fin.close();

        fin.open("C:\\temp\\tsMatrix3.txt", ios::in);
        for(i = 0; i < 11; i++)
        {
                for(int t = 0; t < 11; t++)
                {
                        fin >> number;
                        RetsMatrix3[i][t] = number;
                }
        }
```

```
        fin. close();
        return S_OK;
}

STDMETHODIMP CRe_Order::Re_TourSolution()
{
        //interface methos to solve the Re-Order
        int TourResults[12];
        Find_ReTour(TourResults);
        return S_OK;
}
```

Appendix 3

ArcInfo 8 VBA macro code

VBA module routines

```
Private m_ipPoints As esriCore.IPointCollection
Private m_ipPathFinder As NetObjVB6.PathFinder
Private m_ipTDTSP As TDTSPLib.TSP_alg
Private m_ipReOpt As TDTSPLib.Re_Opt
Private tsMatrix1(11, 11) As Double
Private tsMatrix2(11, 11) As Double
Private tsMatrix3(11, 11) As Double

Sub RunMyUserForm()
'get information for updating the TSP tour
  MyUserForm.Show vbModeless
End Sub


Private Sub UIButtonControl1_Click()
'Build TDTSP travel speed matrices
  Dim ipMxDoc As IMxDocument
  Dim ipMap As IMap
  Dim ipLayer As ILayer
  Dim ipFeatureLayer As IFeatureLayer, ipFeatLayer2 As IFeatureLayer
  Dim ipFeatureCls As IFeatureClass
  Dim ipFeatCls1 As IFeatureClass, ipFeatCls2 As IFeatureClass, ipFeatCls3 As
IFeatureClass
  Dim ipFDB As IFeatureDataset
  Dim ipPolyResult As IPolyline
  Dim i As Integer, j As Integer, k As Integer
  Dim tName1 As String, tName2 As String, tName3 As String

  tName1 = "tourlinks1"
  tName2 = "tourlinks2"
  tName3 = "tourlinks3"

  If m_ipPathFinder Is Nothing Then
    Set m_ipPathFinder = New NetObjVB6.PathFinder
    Set ipMxDoc = ThisDocument
    Set ipMap = ipMxDoc.FocusMap

    Debug.Assert ipMap.LayerCount > 0
```

```
Set ipLayer = ipMap.Layer(0)
Set ipFeatureLayer = ipLayer

'sets ipFeatureCls to Nodes-To-Visit
Set ipFeatureCls = ipFeatureLayer.FeatureClass
MsgBox "Feature Class:" + ipFeatureCls.AliasName

Set ipFDB = ipFeatureLayer.FeatureClass.FeatureDataset

Set m_ipPathFinder.Map = ipMap

m_ipPathFinder.OpenFeatureDatasetNetwork ipFDB

For i = 0 To ipMap.LayerCount - 1
   If ipMap.Layer(i).Name = tName1 Then
      'set ipFeatCls1 to TourLinks_1
      Set ipFeatLayer2 = ipMap.Layer(i)
      Set ipFeatCls1 = ipFeatLayer2.FeatureClass
      MsgBox "Feature Class1:" + ipFeatCls1.AliasName
   End If
   If ipMap.Layer(i).Name = tName2 Then
      'set ipFeatCls2 to TourLinks_2
      Set ipFeatLayer2 = ipMap.Layer(i)
      Set ipFeatCls2 = ipFeatLayer2.FeatureClass
      MsgBox "Feature Class2:" + ipFeatCls2.AliasName
   End If
   If ipMap.Layer(i).Name = tName3 Then
      'set ipFeatCls3 to TourLinks_3
      Set ipFeatLayer2 = ipMap.Layer(i)
      Set ipFeatCls3 = ipFeatLayer2.FeatureClass
      MsgBox "Feature Class3:" + ipFeatCls3.AliasName
   End If
 Next i
End If

Dim ipOFeat As IFeature
Dim ipDFeat As IFeature
Dim ipOPoint As IPoint
Dim ipDPoint As IPoint
Dim Rcount As Integer, Ccount As Integer
Dim ipNewFeature As IFeature
Dim pFlds As IFields

Dim fs, a, b, c
Set fs = CreateObject("Scripting.FileSystemObject")
Set a = fs.CreateTextFile("c:\temp\tsMatrix1.txt", True)
Set b = fs.CreateTextFile("c:\temp\tsMatrix2.txt", True)
Set c = fs.CreateTextFile("c:\temp\tsMatrix3.txt", True)
```

```
Rcount = 0
Ccount = 0

If m_ipPoints Is Nothing Then
    Set m_ipPoints = New esriCore.Multipoint
    Set m_ipPathFinder.StopPoints = m_ipPoints
End If
Dim linkcount As Integer
linkcount = 1

' get the workspace and start editing
Dim pDataset As IDataset
Set pDataset = ipFeatCls1
Dim pWorkspace As IWorkspace
Set pWorkspace = pDataset.Workspace
Dim pWorkspaceEdit As IWorkspaceEdit
Set pWorkspaceEdit = pWorkspace
pWorkspaceEdit.StartEditing False
pWorkspaceEdit.StartEditOperation

For i = 2 To 12
  Set ipOFeat = ipFeatureCls.GetFeature(i)
  Set ipOPoint = ipOFeat.Shape
  For j = 2 To 12
     Set ipDFeat = ipFeatureCls.GetFeature(j)
     Set ipDPoint = ipDFeat.Shape
     'MsgBox CStr(i) + CStr(j)
     If m_ipPoints Is Nothing Then
        Set m_ipPoints = New esriCore.Multipoint
        Set m_ipPathFinder.StopPoints = m_ipPoints
     End If
     m_ipPoints.AddPoint ipOPoint
     m_ipPoints.AddPoint ipDPoint

     Debug.Assert m_ipPoints.PointCount > 1
     'MsgBox "# of Points" + CStr(m_ipPoints.PointCount)

     m_ipPathFinder.SolvePath "TimePd1"
     'MsgBox "Drive Time: " + CStr(m_ipPathFinder.PathCost)
     Set ipPolyResult = m_ipPathFinder.PathPolyLine
     Set ipNewFeature = ipFeatCls1.CreateFeature
     Set ipNewFeature.Shape = ipPolyResult
     Set pFlds = ipNewFeature.Fields
     For k = 0 To pFlds.FieldCount - 1
        If (pFlds.Field(k).Name = "MyID") Then
           ipNewFeature.Value(k) = linkcount
        End If
        If (pFlds.Field(k).Name = "To_Node") Then
           ipNewFeature.Value(k) = j
```

```
    End If
Next k
ipNewFeature.Store

tsMatrix1(Rcount, Ccount) = m_ipPathFinder.PathCost
a.Write CStr(tsMatrix1(Rcount, Ccount))
a.Write (" ")

m_ipPathFinder.SolvePath "TimePd2"
Set ipPolyResult = m_ipPathFinder.PathPolyLine
Set ipNewFeature = ipFeatCls2.CreateFeature
Set ipNewFeature.Shape = ipPolyResult
Set pFlds = ipNewFeature.Fields
For k = 0 To pFlds.FieldCount - 1
    If (pFlds.Field(k).Name = "MyID") Then
        ipNewFeature.Value(k) = linkcount
    End If
    If (pFlds.Field(k).Name = "To_Node") Then
        ipNewFeature.Value(k) = j
    End If
Next k
ipNewFeature.Store

tsMatrix2(Rcount, Ccount) = m_ipPathFinder.PathCost
b.Write CStr(tsMatrix2(Rcount, Ccount))
b.Write (" ")

m_ipPathFinder.SolvePath "TimePd3"
Set ipPolyResult = m_ipPathFinder.PathPolyLine
Set ipNewFeature = ipFeatCls3.CreateFeature
Set ipNewFeature.Shape = ipPolyResult
Set pFlds = ipNewFeature.Fields
For k = 0 To pFlds.FieldCount - 1
    If (pFlds.Field(k).Name = "MyID") Then
        ipNewFeature.Value(k) = linkcount
    End If
    If (pFlds.Field(k).Name = "To_Node") Then
        ipNewFeature.Value(k) = j
    End If
Next k
ipNewFeature.Store

tsMatrix3(Rcount, Ccount) = m_ipPathFinder.PathCost
c.Write CStr(tsMatrix3(Rcount, Ccount))
c.Write (" ")

Ccount = Ccount + 1
linkcount = linkcount + 1
Set m_ipPoints = Nothing
```

```
    Next j
    Rcount = Rcount + 1
    Ccount = 0
    a.WriteBlankLines (1)
    b.WriteBlankLines (1)
    c.WriteBlankLines (1)

  Next i
  a.Close

  pWorkspaceEdit.StopEditOperation
  pWorkspaceEdit.StopEditing True

  MsgBox "DONE building shortest paths"
End Sub


Private Function UIButtonControl1_ToolTip() As String
    UIButtonControl1_ToolTip = "Build Shortest Paths"
End Function


Private Sub UIButtonControl2_Click()
'Display the TDTSP tour

  Dim ipPolyResult As IPolyline
  Dim ipMap As IMap
  Dim ipFeatLayer As IFeatureLayer
  Dim ipFeatureCls1 As IFeatureClass
  Dim ipFeatureCls2 As IFeatureClass, ipFeatureCls3 As IFeatureClass
  Dim ipMxDoc As IMxDocument
  Dim ipGraphicsContainer As IGraphicsContainer
  Dim ipElemet As IElement
  Dim pScreenDisplay As IScreenDisplay
  Dim ipLineSymbol As ILineSymbol
  Dim m_ipClipEnv As esriCore.Envelope
  Dim node, ntotal As Integer
  Dim node1, node2 As Integer
  Dim total, t As Integer
  Dim i, j As Integer

  Set ipMxDoc = ThisDocument
  Set pScreenDisplay = ipMxDoc.ActiveView.ScreenDisplay
  Set ipMap = ipMxDoc.FocusMap

  Set ipLineSymbol = New CartographicLineSymbol
  ipLineSymbol.Width = 5
```

```
Open "c:\temp\tdtsptour.txt" For Input As #1

Dim pFields As IFields
Dim LinkID, lIndex As Long
Dim lIndex2 As Long, lIndex3 As Long
Dim tIndex As Long, tIndex2 As Long, tIndex3 As Long
Dim lID As Long
lID = 1

Dim pReInSelect As IFeatureSelection, pReInSelect2 As IFeatureSelection, pReInSelect3
As IFeatureSelection
Dim pFC As IFeatureCursor
Dim pTempFeat As IFeature

For j = 0 To ipMap.LayerCount - 1
    If (ipMap.Layer(j).Name = "tourlinks1") Then
        Set ipFeatLayer = ipMap.Layer(j)
        Set ipFeatureCls1 = ipFeatLayer.FeatureClass
        Set pReInSelect = ipFeatLayer
        'MsgBox ipFeatureCls1.AliasName
    End If
    If (ipMap.Layer(j).Name = "tourlinks2") Then
        Set ipFeatLayer = ipMap.Layer(j)
        Set ipFeatureCls2 = ipFeatLayer.FeatureClass
        Set pReInSelect2 = ipFeatLayer
        'MsgBox ipFeatureCls2.AliasName
    End If
    If (ipMap.Layer(j).Name = "tourlinks3") Then
        Set ipFeatLayer = ipMap.Layer(j)
        Set ipFeatureCls3 = ipFeatLayer.FeatureClass
        Set pReInSelect3 = ipFeatLayer
        'MsgBox ipFeatureCls3.AliasName
    End If
Next j

' get the workspace and start editing
Dim pDataset As IDataset
Set pDataset = ipFeatureCls1
Dim pWorkspace As IWorkspace
Set pWorkspace = pDataset.Workspace
Dim pWorkspaceEdit As IWorkspaceEdit
Set pWorkspaceEdit = pWorkspace
pWorkspaceEdit.StartEditing False
pWorkspaceEdit.StartEditOperation

Set pFields = ipFeatureCls1.Fields
lIndex = pFields.FindField("Position_")
'For j = 0 To ipFeatureCls1.FeatureCount - 1
```

```
tIndex = pFields.FindField("Acc_Time")
Set pFields = ipFeatureCls2.Fields
lIndex2 = pFields.FindField("Position_")
tIndex2 = pFields.FindField("Acc_Time")
Set pFields = ipFeatureCls3.Fields
lIndex3 = pFields.FindField("Position_")
tIndex3 = pFields.FindField("Acc_Time")

Set ipSelect = pReInSelect.SelectionSet
ipSelect.Search Nothing, False, pFC
MsgBox ipSelect.count
If (ipSelect.count > 0) Then
 'MsgBox "here"
   Set pTempFeat = pFC.NextFeature
   Do Until (pTempFeat Is Nothing)
     pTempFeat.Value(lIndex) = Null
     pTempFeat.Value(tIndex) = Null
     pTempFeat.Store
     Set pTempFeat = pFC.NextFeature
  Loop
 End If
 Set ipSelect = pReInSelect2.SelectionSet
ipSelect.Search Nothing, False, pFC
 If (ipSelect.count > 0) Then
 'MsgBox "here2"
   Set pTempFeat = pFC.NextFeature
   Do Until (pTempFeat Is Nothing)
     pTempFeat.Value(lIndex) = Null
     pTempFeat.Value(tIndex) = Null
     pTempFeat.Store
     Set pTempFeat = pFC.NextFeature
  Loop
 End If
Set ipSelect = pReInSelect3.SelectionSet
ipSelect.Search Nothing, False, pFC
 If (ipSelect.count > 0) Then
 'MsgBox "here3"
   Set pTempFeat = pFC.NextFeature
   Do Until (pTempFeat Is Nothing)
     pTempFeat.Value(lIndex) = Null
     pTempFeat.Value(tIndex) = Null
     pTempFeat.Store
     Set pTempFeat = pFC.NextFeature
  Loop
 End If

 ipMap.ClearSelection

Dim ipSelectFeat As IFeature
```

```
pScreenDisplay.StartDrawing 0, esriNoScreenCache
pScreenDisplay.SetSymbol ipLineSymbol
Dim count As Integer
count = 1

Do While Not EOF(1)
 Input #1, node1, ntotal
 t = ntotal
 For i = 1 To 11
    Input #1, node2, ntotal
    t = t + ntotal
    'MsgBox "t total: " + CStr(t)
    If (node1 = 0) Then
       LinkID = lID + node2
       'LinkID = node2
       If (t < 211) Then
          Set ipSelectFeat = ipFeatureCls1.GetFeature(LinkID)
       ElseIf (t > 421) Then
          Set ipSelectFeat = ipFeatureCls3.GetFeature(LinkID)
       Else
         Set ipSelectFeat = ipFeatureCls2.GetFeature(LinkID)
       End If
    End If
    If (node1 = 1) Then
       LinkID = lID + node2 + 11
       'LinkID = node2 + 11
       If (t < 211) Then
          Set ipSelectFeat = ipFeatureCls1.GetFeature(LinkID)
       ElseIf (t > 421) Then
          Set ipSelectFeat = ipFeatureCls3.GetFeature(LinkID)
       Else
          Set ipSelectFeat = ipFeatureCls2.GetFeature(LinkID)
       End If
    End If
    If (node1 = 2) Then
       LinkID = lID + node2 + 22
       'LinkID = node2 + 22
       If (t < 211) Then
          Set ipSelectFeat = ipFeatureCls1.GetFeature(LinkID)
       ElseIf (t > 421) Then
          Set ipSelectFeat = ipFeatureCls3.GetFeature(LinkID)
       Else
          Set ipSelectFeat = ipFeatureCls2.GetFeature(LinkID)
       End If
    End If
    If (node1 = 3) Then
       LinkID = lID + node2 + 33
       'LinkID = node2 + 33
       If (t < 211) Then
```

```
        Set ipSelectFeat = ipFeatureCls1.GetFeature(LinkID)
    ElseIf (t > 421) Then
        Set ipSelectFeat = ipFeatureCls3.GetFeature(LinkID)
    Else
        Set ipSelectFeat = ipFeatureCls2.GetFeature(LinkID)
    End If
End If
If (node1 = 4) Then
    LinkID = lID + node2 + 44
    'LinkID = node2 + 44
    If (t < 211) Then
        Set ipSelectFeat = ipFeatureCls1.GetFeature(LinkID)
    ElseIf (t > 421) Then
        Set ipSelectFeat = ipFeatureCls3.GetFeature(LinkID)
    Else
        Set ipSelectFeat = ipFeatureCls2.GetFeature(LinkID)
    End If
End If
If (node1 = 5) Then
    LinkID = lID + node2 + 55
    'LinkID = node2 + 55
    If (t < 211) Then
        Set ipSelectFeat = ipFeatureCls1.GetFeature(LinkID)
    ElseIf (t > 421) Then
        Set ipSelectFeat = ipFeatureCls3.GetFeature(LinkID)
    Else
        Set ipSelectFeat = ipFeatureCls2.GetFeature(LinkID)
    End If
End If
If (node1 = 6) Then
    LinkID = lID + node2 + 66
    'LinkID = node2 + 66
    If (t < 211) Then
        Set ipSelectFeat = ipFeatureCls1.GetFeature(LinkID)
    ElseIf (t > 421) Then
        Set ipSelectFeat = ipFeatureCls3.GetFeature(LinkID)
    Else
        Set ipSelectFeat = ipFeatureCls2.GetFeature(LinkID)
    End If
End If
If (node1 = 7) Then
    LinkID = lID + node2 + 77
    'LinkID = node2 + 77
    If (t < 211) Then
        Set ipSelectFeat = ipFeatureCls1.GetFeature(LinkID)
    ElseIf (t > 421) Then
        Set ipSelectFeat = ipFeatureCls3.GetFeature(LinkID)
    Else
        Set ipSelectFeat = ipFeatureCls2.GetFeature(LinkID)
```

```
    End If
  End If
  If (node1 = 8) Then
    LinkID = lID + node2 + 88
    'LinkID = node2 + 88
    If (t < 211) Then
       Set ipSelectFeat = ipFeatureCls1.GetFeature(LinkID)
    ElseIf (t > 421) Then
      Set ipSelectFeat = ipFeatureCls3.GetFeature(LinkID)
    Else
      Set ipSelectFeat = ipFeatureCls2.GetFeature(LinkID)
    End If
  End If
  If (node1 = 9) Then
    LinkID = lID + node2 + 99
    'LinkID = node2 + 99
    If (t < 211) Then
       Set ipSelectFeat = ipFeatureCls1.GetFeature(LinkID)
    ElseIf (t > 421) Then
      Set ipSelectFeat = ipFeatureCls3.GetFeature(LinkID)
    Else
      Set ipSelectFeat = ipFeatureCls2.GetFeature(LinkID)
    End If
  End If
  If (node1 = 10) Then
      LinkID = lID + node2 + 110
      'LinkID = node2 + 110
    If (t < 211) Then
       Set ipSelectFeat = ipFeatureCls1.GetFeature(LinkID)
    ElseIf (t > 421) Then
      Set ipSelectFeat = ipFeatureCls3.GetFeature(LinkID)
    Else
      Set ipSelectFeat = ipFeatureCls2.GetFeature(LinkID)
    End If
  End If

  Set ipPolyResult = ipSelectFeat.Shape
  If (t < 211) Then
    ipSelectFeat.Value(lIndex) = count
    ipSelectFeat.Value(tIndex) = t
    ipSelectFeat.Store
  ElseIf (t > 421) Then
    ipSelectFeat.Value(lIndex3) = count
    ipSelectFeat.Value(tIndex3) = t
    ipSelectFeat.Store
  Else
    ipSelectFeat.Value(lIndex2) = count
    ipSelectFeat.Value(tIndex2) = t
    ipSelectFeat.Store
```

```
        End If

        pScreenDisplay.DrawPolyline ipPolyResult
        Set m_ipClipEnv = pScreenDisplay.ClipEnvelope
        MsgBox "Tour Link number: " + CStr(count)
        node1 = node2
        count = count + 1
      Next i
      Input #1, total
      'MsgBox "t total" + CStr(t)
      MsgBox "Total: " + CStr(total)
    Loop
    pWorkspaceEdit.StopEditOperation
    pWorkspaceEdit.StopEditing True
    Close #1    ' Close file
    pScreenDisplay.FinishDrawing

End Sub


Private Function UIButtonControl2_ToolTip() As String
    UIButtonControl2_ToolTip = "Display TDTSP tour"
End Function


Private Sub UIButtonControl3_Click()
    'Set m_Update = New ClassGet
    'MsgBox m_Update.AccidentName
    Set m_ipTDTSP = New TDTSPLib.TSP_alg
    m_ipTDTSP.import_Matrix
    m_ipTDTSP.TourSolution
    MsgBox "TDTSP Routing Algorithm Executed"
End Sub

Private Sub UIButtonControl4_Click()
'RunReOrder heuristic
    Set m_ipReOpt = New TDTSPLib.Re_Opt
    m_ipReOpt.import_Re_Matrix
    MsgBox "read in ts"
    m_ipReOpt.Re_TourSolution
    MsgBox "Re-OPt Routing Algorithm Executed"
End Sub

'ShortestPath tool
Private Sub UIToolControl1_DblClick()
Dim ipPolyResult As IPolyline
  Dim ipMxDoc As IMxDocument
  Dim ipGraphicsContainer As IGraphicsContainer
  Dim ipElemet As IElement
```

```
  Dim pScreenDisplay As IScreenDisplay
  Dim ipLineSymbol As ILineSymbol
  Dim m_ipClipEnv As esriCore.Envelope

  On Error GoTo CleanUp

  Debug.Assert m_ipPoints.PointCount > 1
  ' change this to reflect your weight name
  m_ipPathFinder.SolvePath "Miles"

  Set ipPolyResult = m_ipPathFinder.PathPolyLine
  Set ipMxDoc = ThisDocument
  Set pScreenDisplay = ipMxDoc.ActiveView.ScreenDisplay

  Set ipLineSymbol = New CartographicLineSymbol
  ipLineSymbol.Width = 5

  pScreenDisplay.StartDrawing 0, esriNoScreenCache

  pScreenDisplay.SetSymbol ipLineSymbol
  pScreenDisplay.DrawPolyline ipPolyResult
  Set m_ipClipEnv = pScreenDisplay.ClipEnvelope
  pScreenDisplay.FinishDrawing

  MsgBox "Drive Time: " + CStr(m_ipPathFinder.PathCost)

CleanUp:

  Set m_ipPoints = Nothing ' clear it
End Sub


'shortest path tool
Private Sub UIToolControl1_MouseDown(ByVal button As Long, ByVal shift As Long,
ByVal X As Long, ByVal Y As Long)
  Dim pMxApp As IMxApplication
  Dim ipNew As IPoint

  If m_ipPoints Is Nothing Then
    Set m_ipPoints = New esriCore.Multipoint
    Set m_ipPathFinder.StopPoints = m_ipPoints
  End If

  Set pMxApp = Application
  Set ipNew = pMxApp.Display.DisplayTransformation.ToMapPoint(X, Y)

  m_ipPoints.AddPoint ipNew
End Sub
```

```
Private Sub UIToolControl1_Select() 'shortest path tool
  Dim ipMxDoc As IMxDocument
  Dim ipMap As IMap
  Dim ipLayer As ILayer
  Dim ipFeatureLayer As IFeatureLayer
  Dim ipFDB As IFeatureDataset

  If m_ipPathFinder Is Nothing Then
    Set m_ipPathFinder = New NetObjVB6.PathFinder

    Set ipMxDoc = ThisDocument
    Set ipMap = ipMxDoc.FocusMap

    Debug.Assert ipMap.LayerCount > 0
    Set ipLayer = ipMap.Layer(0)
    Set ipFeatureLayer = ipLayer

    Set ipFDB = ipFeatureLayer.FeatureClass.FeatureDataset
    Set m_ipPathFinder.Map = ipMap

    m_ipPathFinder.OpenFeatureDatasetNetwork ipFDB

  End If
End Sub


Private Function UIToolControl1_ToolTip() As String
    UIToolControl1_ToolTip = "Shortest Path"
End Function
```

MyUserUpdateForm Macro Code

```
Private pMyUpdate As GetUpdates
Private m_pMyUpdate2 As GetUpdates

Private Sub Cancel_dButton1_Click()
    Unload MyUserForm
End Sub

Private Sub OK_Button_Click()
    Dim timetxt As String
    Dim pMxDoc As IMxDocument
    Set pMxDoc = ThisDocument
    Dim pMap As IMap
    Set pMap = pMxDoc.FocusMap
```

```
    Set pMyUpdate = New GetUpdates

    pMyUpdate.TourName = TourText.Text
    pMyUpdate.TourName2 = TourText2.Text
    pMyUpdate.TourName3 = TourText3.Text
    pMyUpdate.AccidentName = AccidText.Text
    pMyUpdate.Time = TimeText.Text
    Set pMyUpdate.MyMxDoc = pMxDoc 'this launches the FindSelectedAccidents class
method
    Unload MyUserForm

End Sub
```

GetUpdates Class (implemented by *MyUserUpdateForm*)

```
Private m_TourName As String, m_TourName2 As String, m_TourName3 As String
Private m_AccidName As String
Private m_Time As Double
Private m_MyMxDoc As IMxDocument

Private Type LinksToReOpt
      LinkID As Long
      ToNode As Long
      Delay As Double
End Type

Public Property Get TourName() As String
   TourName = m_TourName
End Property

Public Property Let TourName(ByVal TourName As String)
   m_TourName = TourName
End Property
Public Property Get TourName2() As String
   TourName2 = m_TourName2
End Property

Public Property Let TourName2(ByVal TourName2 As String)
   m_TourName2 = TourName2
End Property

Public Property Get TourName3() As String
   TourName3 = m_TourName3
End Property

Public Property Let TourName3(ByVal TourName3 As String)
   m_TourName3 = TourName3
```

```
End Property

Public Property Get AccidentName() As String
   AccidentName = m_AccidName
End Property

Public Property Let AccidentName(ByVal AccidentName As String)
   m_AccidName = AccidentName
End Property

Public Property Get Time() As Double
   Time = m_Time
End Property

Public Property Let Time(ByVal Time As Double)
   m_Time = Time
End Property

Public Property Set MyMxDoc(MyMxDoc As IMxDocument)
   Set m_MyMxDoc = MyMxDoc
   MsgBox m_MyMxDoc.Maps.count
   FindSelectedAccidents
End Property

Public Sub FindSelectedAccidents()
'function called by MyUserUpdateForm
   Dim pMap As IMap
   Dim pAFeatLayer As IFeatureLayer
   Dim pAFeatSelection As IFeatureSelection
   Dim pAFeatCursor As IFeatureCursor
   Dim pAFeatCursor2 As IFeatureCursor, pAFeatCursor3 As IFeatureCursor
   Dim pAFeature As IFeature
   Dim pTFeatLayer As IFeatureLayer
   Dim pTFeatLayer2 As IFeatureLayer, pTFeatLayer3 As IFeatureLayer
   Dim pTFeatSelection As IFeatureSelection
   Dim pTFeatCursor As IFeatureCursor
   Dim pTFeature As IFeature
   Dim pFields As IFields
   Dim i As Long, temp As Long
   Dim Delay As Double
   Dim LinkID As Long, lIndex As Long
   Dim accTime As Long, tIndex As Long
   Dim ToNode As Long
   Dim pSpatialFilter As ISpatialFilter, pSpatialFilter2 As IspatialFilter
   Dim pSpatialFilter3 As ISpatialFilter

'get the Map object to get the feature layers
   Set pMap = m_MyMxDoc.FocusMap
```

```
For i = 0 To pMap.LayerCount – 1
  'find the TourLInk classes and Accident feature class that were inputed in the form
  If (pMap.Layer(i).Name = m_AccidName) Then
    Set pAFeatLayer = pMap.Layer(i)
  End If
  If (pMap.Layer(i).Name = m_TourName) Then
    Set pTFeatLayer = pMap.Layer(i)
  End If
  If (pMap.Layer(i).Name = m_TourName2) Then
    Set pTFeatLayer2 = pMap.Layer(i)
  End If
  If (pMap.Layer(i).Name = m_TourName3) Then
    Set pTFeatLayer3 = pMap.Layer(i)
  End If
Next i
'MsgBox pTFeatLayer.Name
' MsgBox pTFeatLayer2.Name
'MsgBox pTFeatLayer3.Name
Dim fs, a
'open the ReOrder file for output
Set fs = CreateObject("Scripting.FileSystemObject")
Set a = fs.CreateTextFile("c:\temp\ReOrder.txt", True)

If (m_Time < 180) Then
'then tour not past this TourLinks1 yet so must check if any links selected
  Set pTFeatSelection = pTFeatLayer
  If (pTFeatSelection.SelectionSet.count <> 0) Then
  MsgBox "make it past 180 <>)"
    pTFeatSelection.SelectionSet.Search Nothing, True, pTFeatCursor
   For i = 0 To pTFeatSelection.SelectionSet.count - 1
    Set pTFeature = pTFeatCursor.NextFeature
    Set pFields = pTFeatLayer.FeatureClass.Fields
    tIndex = pFields.FindField("Acc_Time")
    accTime = pTFeature.Value(tIndex)
    MsgBox accTime
    MsgBox "count" + CStr(i)
    If (accTime > m_Time) Then
    'then have not reached this link
      temp = m_Time + 120
      'MsgBox temp
      If (temp > accTime) Then
        lIndex = pFields.FindField("Position_")
        LinkID = pTFeature.Value(lIndex)
        MsgBox "make it past accTime 180"
        If (LinkID < 9) Then
          MsgBox "Need to Re-Opt" + CStr(LinkID)
          lIndex = pFields.FindField("To_Node")
          ToNode = pTFeature.Value(lIndex)
          a.Write CStr(LinkID)
```

```
                a.Write (" ")
                a.Write CStr(ToNode)
                a.Write (" ")
                'now get associated accident delay value
                 Set pSpatialFilter = New SpatialFilter
                 Set pSpatialFilter.Geometry = pTFeature.Shape
                 MsgBox pTFeature.OID
                 pSpatialFilter.SpatialRel = esriSpatialRelIntersects
                 pSpatialFilter.GeometryField = pAFeatLayer.FeatureClass.ShapeFieldName
                 Set pAFeatCursor = pAFeatLayer.Search(pSpatialFilter, True)
                 Set pAFeature = pAFeatCursor.NextFeature
                 Set pFields = pAFeatLayer.FeatureClass.Fields
                 lIndex = pFields.FindField("Delay_Value")
                 Delay = pAFeature.Value(lIndex)
                 MsgBox "delay" + CStr(Delay)
                 a.Write CStr(Delay)
                 a.Write (" ")
                 lIndex = pFields.FindField("OBJECTID")
                 MsgBox pAFeature.Value(lIndex)
               End If '< 9
          End If 'accTime
        End If 'accTime
      Next i
  End If '<>0
  End If
 If (m_Time > 600) Then
'then tour not past this TourLinks3 yet so must check if any links selected

   Set pTFeatSelection = pTFeatLayer3
   If (pTFeatSelection.SelectionSet.count <> 0) Then
   MsgBox "make it past 600 <>)"
      pTFeatSelection.SelectionSet.Search Nothing, True, pTFeatCursor
     For i = 0 To pTFeatSelection.SelectionSet.count - 1
      Set pTFeature = pTFeatCursor.NextFeature
      Set pFields = pTFeatLayer3.FeatureClass.Fields
      tIndex = pFields.FindField("Acc_Time")
      accTime = pTFeature.Value(tIndex)
      MsgBox accTime
      If (accTime > m_Time) Then
      'then have not reached this link
         temp = m_Time + 120
         'MsgBox temp
         If (temp > accTime) Then
           lIndex = pFields.FindField("Position_")
           LinkID = pTFeature.Value(lIndex)
           MsgBox "make it past accTime 600"
           If (LinkID < 9) Then
              MsgBox "Need to Re-Opt" + CStr(LinkID)
              lIndex = pFields.FindField("To_Node")
```

```
            ToNode = pTFeature.Value(lIndex)
            a.Write CStr(LinkID)
            a.Write (" ")
            a.Write CStr(ToNode)
            a.Write (" ")
            'now get associated accident delay value
            Set pSpatialFilter3 = New SpatialFilter
             Set pSpatialFilter3.Geometry = pTFeature.Shape
             pSpatialFilter3.SpatialRel = esriSpatialRelIntersects
             pSpatialFilter3.GeometryField = pAFeatLayer.FeatureClass.ShapeFieldName
             Set pAFeatCursor3 = pAFeatLayer.Search(pSpatialFilter3, True)
             Set pAFeature = pAFeatCursor3.NextFeature
             Set pFields = pAFeatLayer.FeatureClass.Fields
             lIndex = pFields.FindField("Delay_Value")
             Delay = pAFeature.Value(lIndex)
             MsgBox "delay" + CStr(Delay)
             a.Write CStr(Delay)
             a.Write (" ")
             lIndex = pFields.FindField("OBJECTID")
             MsgBox pAFeature.Value(lIndex)
          End If '< 9
        End If 'accTime
      End If 'accTime
    Next i
End If '<>0
End If
If (m_Time < 480) Then
 'then tour not past this TourLinks3 yet so must check if any links selected

 Set pTFeatSelection = pTFeatLayer2
 If (pTFeatSelection.SelectionSet.count <> 0) Then
 MsgBox "make it past * <>)"
    pTFeatSelection.SelectionSet.Search Nothing, True, pTFeatCursor
  For i = 0 To pTFeatSelection.SelectionSet.count - 1
    Set pTFeature = pTFeatCursor.NextFeature
    Set pFields = pTFeatLayer2.FeatureClass.Fields
    tIndex = pFields.FindField("Acc_Time")
    accTime = pTFeature.Value(tIndex)
    MsgBox accTime
    If (accTime > m_Time) Then
    'then have not reached this link
       temp = m_Time + 120
       'MsgBox temp
       If (temp > accTime) Then
         lIndex = pFields.FindField("Position_")
         LinkID = pTFeature.Value(lIndex)
         MsgBox "make it past accTime *"
         If (LinkID < 9) Then
            MsgBox "Need to Re-Opt" + CStr(LinkID)
```

```
                lIndex = pFields.FindField("To_Node")
                ToNode = pTFeature.Value(lIndex)
                a.Write CStr(LinkID)
                a.Write ("  ")
                a.Write CStr(ToNode)
                a.Write ("  ")
                'now get associated accident delay value
                 Set pSpatialFilter2 = New SpatialFilter
                 Set pSpatialFilter2.Geometry = pTFeature.Shape
                 MsgBox pTFeature.OID
                 pSpatialFilter2.SpatialRel = esriSpatialRelIntersects
                 pSpatialFilter2.GeometryField = pAFeatLayer.FeatureClass.ShapeFieldName
                 Set pAFeatCursor2 = pAFeatLayer.Search(pSpatialFilter2, True)
                 Set pAFeature = pAFeatCursor2.NextFeature
                 Set pFields = pAFeatLayer.FeatureClass.Fields
                 lIndex = pFields.FindField("Delay_Value")
                 Delay = pAFeature.Value(lIndex)
                 MsgBox "delay" + CStr(Delay)
                 a.Write CStr(Delay)
                 a.Write ("  ")
                 lIndex = pFields.FindField("OBJECTID")
                 MsgBox pAFeature.Value(lIndex)
              End If '< 9
            End If 'accTime
          End If 'accTime
        Next i
      End If '<>0
    End If

    a.Write CStr(m_Time)
    a.Write ("  ")
    a.Close

End Sub
```

Vita

Melinda R. Vazzana was born in Washington D.C. on March 27, 1977. She received her B.A. in Liberal Science with a concentration in Anthropology and minor in Geography at Salisbury State University Salisbury, MD in May 1999. She will complete her M.S. in Geography, with a concentration in GIS-Transportation, at the University of Tennessee, Knoxville in May 2003.

7811 0207 56
11/05/03 MRB