Fall 2013

Probabilistic Modeling of Erroneous Human Response to In-Vehicle Route Guidance Systems: A Domain Decomposition-Based Algorithm

Abdullah Al Farooq
Old Dominion University

PROBABILISTIC MODELING OF ERRONEOUS HUMAN RESPONSE TO IN-VEHICLE ROUTE GUIDANCE SYSTEMS: A DOMAIN DECOMPOSITION-BASED ALGORITHM

by

Abdullah Al Farooq
B.Sc. February 2011, Bangladesh University of Engineering & Technology

A Thesis Submitted to the Faculty of
Old Dominion University in Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

MODELING AND SIMULATION

OLD DOMINION UNIVERSITY
December 2013

Approved by:

____
ManWo Ng (Director)

____
Duc T. Nguyen (Member)

____
Jiang Li (Member)

ABSTRACT

PROBABILISTIC MODELING OF ERRONEOUS HUMAN RESPONSE TO IN-VEHICLE ROUTE GUIDANCE SYSTEM: A DOMAIN DECOMPOSITION-BASED ALGORITHM

Abdullah Al Farooq
Old Dominion University, 2013
Director: Dr. ManWo Ng

Drivers are generally assumed to follow the shortest route to their destinations prescribed by in-vehicle route navigation systems without mistake. However, it is not uncommon that drivers make mistakes when there are complex intersections along the route to miss turns. Such mistakes would result in following a longer route than the optimal one. In this thesis, research has been conducted to analyze the effects of the number of intersections in the shortest routes of different O-D (origin-destination) pairs. While different formulations can be employed to describe such “driving with error” model, a Domain Decomposition (DD) partitioning algorithm has been developed for route guidance systems in order to recommend optimal routes to the drivers. A numerical comparison among different solution approaches has been conducted in this thesis.

This thesis is dedicated to my family.

ACKNOWLEDGMENTS

I would like to thank all of the people that have made this thesis possible. First and foremost, I would like to thank my advisor, Dr. ManWo Ng for letting me enter into the world of transportation. I am also grateful for his guidance, constant support, and many suggestions during this research. My gratitude also goes to Dr. Duc T. Nguyen for continually offering insights and help for this research. I am also thankful to Dr. Jiang Li for taking interest in this research, serving on my committee, and providing me with many useful comments and suggestions.

Outside the committee, I would like to thank Paul Johnson, a Ph.D. student of Civil Engineering, for providing me with very useful materials and insights for this research. Last but not least, I would like to acknowledge my wife, Synthia, for her boundless love and continuous support.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

In-vehicle route navigation systems are becoming increasingly common in modern vehicles. An integral component of these navigation systems is the determination of “optimal” routes (e.g. the fastest) to the desired destination. An implicit assumption that is fundamental to these in-vehicle navigation systems is that the directions provided are followed without mistakes. However, it is probably not hard to recall a situation in which we have missed a turn, despite the directions of a navigation system, especially when driving in an unfamiliar environment. Recently, Ng and Sathasivan [1] relaxed this assumption of error-free human response to in-vehicle navigation systems (henceforth “driving mistake” or “driving error” will be used to denote this type of mistake) and explicitly model the reality that drivers might miss turns, leading to more robust route guidance. To model this, they employed a Markov Decision Process (MDP), e.g. see Puterman [2] and Bertsekas [3]. The resulting MDP was solved as a linear program. The research, conducted in this thesis, is to solve large scale MDPs formulated from real world network accurately and efficiently[1].

1.1 MOTIVATION OF THE THESIS

Whereas the focus in Ng and Sathasivan [1] was on modeling and examining model properties, no solution algorithm, in particular, was discussed. This is the principal observation which motivated this thesis. Two different solution approaches, one is value iteration [4] and the other is policy iteration [2], are presented to solve this problem.

[1] Figures, tables, and references in this thesis were formatted based on the IEEE Transactions and Journals style.

According to the model, described by Ng and Sathasivan [1], as the most interesting – but perhaps extreme – example, it has been noticed that under certain conditions, it is no longer optimal to recommend drivers to take the shortest route. Instead, a longer route (in certain cases even the longest!) becomes optimal. In this thesis, real-world, large scale transportation networks (i.e. Austin, Chicago Regional) will be used to investigate this phenomenon. To this end, a specialized solution algorithm will be developed that is able to solve large-scale instances of the problem.

1.2 SUMMARY OF CONTRIBUTIONS

In this thesis, a robust solution algorithm, a domain decomposition based policy iteration algorithm, is presented to solve the MDP problem associated with driving error. Benefits of the model in terms of travel time savings are investigated via several case studies. A commonly used technique, value iteration, for solving the MDPs and comparing the solution time among different solution approaches is also discussed.

In addition, specific values for error probabilities were assigned according to a defined scale which, in turn, shows logical relationship of error probabilities at different nodes with various numbers of outgoing directions. It should be noted that the goal of this research is not assigning exact error probabilities associated with intersections of a network. Rather, the focus is mainly on developing a solution algorithm which is capable of solving the “driving with error” model with any type of error probabilities.

This thesis seeks to remedy both space inefficiency and time inefficiency for solving large scale MDPs, by modeling solution to remedy the inefficiencies, and by incorporating the solution into a unified model. A novel decomposition algorithm is presented to efficiently solve the MDPs. To this end, the domain decomposition (DD) solver [5] is embedded in direct solution method of policy iteration algorithm to further increase its efficiency. In addition to the enhanced efficiency, the DD algorithm, framed in sparse matrix representation, not only reduces excessive resource consumption, but also facilitates an efficient parallel implementation, which allows the resulting algorithms to scale far beyond single-machine capabilities.

Although the focus is on devising the solution algorithm for navigation systems, it should be noticed that the framework developed in this thesis has potential applications in other domains as well, including the modeling of disobedience to evacuation orders (e.g. see Ng et al. [6]; Ng and Waller [7]) or traveler information systems (e.g. see Yin and Yang [8], Szeto and Lo [9], Toledo and Beinhaker [10], Bertini and Lovell [11]). Moreover, the research conducted in this thesis has implications for travel time reliability (Higatani et al. [12]; Uno et al. [13], Rakha et al. [14]): Variability in travel time is now caused by the failure to follow the intended path, unlike in other studies where demand and supply uncertainties are responsible for travel time unreliability (e.g. see Chen et al. [15], Lo and Tung [16], Lo et al. [17], Sumalee et al. [18], Ng and Waller [19], Ng et al. [20]). Finally, the research presented in this thesis has the potential to reduce unsafe driving behavior (i.e., drivers attempting to make last-minute corrective steering actions), or even lead to greater sustainability through reduced fuel usage.

1.3 OUTLINE OF THE THESIS

The remainder of this thesis is organized as follows. Chapter 2: MODEL REVIEW briefly reviews the MDP under consideration and demonstrates its relevance using by comparing it to the classical shortest path problem [21] using real transportation networks. In Chapter 3: SOLUTION APPROACH, the basic policy iteration algorithm and value iteration algorithms are briefly reviewed to solve Markov Decision Process (MDP) model. Chapter 4: STORAGE SCHEME presents how the transportation network information and MDP model are stored efficiently. Chapter 5: ENHANCED POLICY ITERATION briefly reviews Domain Decomposition (DD) algorithm [22, 23] to show how the algorithm is used to solve the problem. PERFORMANCE COMPARISON AND DISCUSSION (using two different real world networks: Austin Network, and Chicago Regional Network) and comparisons between the classical (without using DD partitioning) approaches, and the approaches, coupled with DD partitioning algorithm, are presented and discussed in Chapter 6. Finally, Chapter 7, CONCLUSION, concludes the thesis by summarizing the main contributions of this work and the identification of potentially worthwhile future research directions.

CHAPTER 2

MODEL REVIEW

This chapter presents introductory materials on how erroneous response to in-vehicle navigation system was modeled [1] as MDP. Basic model properties are discussed in this chapter. This chapter also addresses the need of using this model rather than using the classical one (e.g. classical shortest path algorithm). Numerical examples, from real world networks, are shown to emphasize the critical importance of the model.

2.1 MODEL PROPERTIES

Consider a transportation network G(N, A), with node set N and link set A. Each link (i, j) has an (expected) travel time.$c_{ij} \geq 0$. Let $U(i)$ denote the set of all outgoing links at node i. In case one allows for the possibility of mistakes, when instructed to choose link $u \in U(i)$ at node i, a probability distribution $p_{ij}(u)$ is induced over all nodes j with $(i, j) \in A$. (Of course, $\sum p_{ij}(u) = 1$.) That is, when instructed to go to node j from node i, it is possible that one – by mistake – ends up at node $k \neq j$, where $(i, k) \in A$. (In other words, when instructed to turn right at the next intersection, it now becomes possible that one misses the turn, and goes straight instead.) The instruction $u = (i, j)$ thus induces an immediate expected travel time of $C(i, u) \equiv \sum_j p_{ij}(u)\, c_{ij}$.

The goal, for a navigation system, is then to choose a set of control strategies $u \in U(i)$ at each node i so that the destination D is reached within the shortest expected travel time. Without loss of generality (as in Ng and Sathasivan [1]), it is assumed that the following for the transition probabilities $p_{ij}(u)$: Suppose that a navigation system prescribes us to follow link $u = (i, j)$ at node $i \neq D$. The driver is then assumed to follow this recommendation with a probability of $1 - p_i$ $(0 \leq p_i \leq 1)$, whereas the driver moves to node $k \neq j$ where$(i, k) \in U(i)$ with probability $\frac{p_i}{|U(i)|-1}$ (here $|B|$ is used to denote the cardinality of the set B). That is, when making a mistake, it is assumed that all wrong turns are equally likely. For the destination node, we simply define $C(D, u) = 0, \forall u \in U(D)$.

Let $e(i) = 1,2, \ldots., n$ denote the minimum expected travel time from node *i* to the destination node. (Note that by definition $e(D) = 0$, so that the cost label can be omitted from the analysis.) It can then be shown that $e(i)$ satisfies Bellman's equations (cf. Puterman [2]; Bertsekas [3]):

$$e(i) = \min_{u \in U(i)} [\, C(i, u) + \textstyle\sum_j p_{ij}(u) e(j) \,] \qquad (1)$$

For more details on this MDP, the reader is referred to Ng and Sathasivan [1].

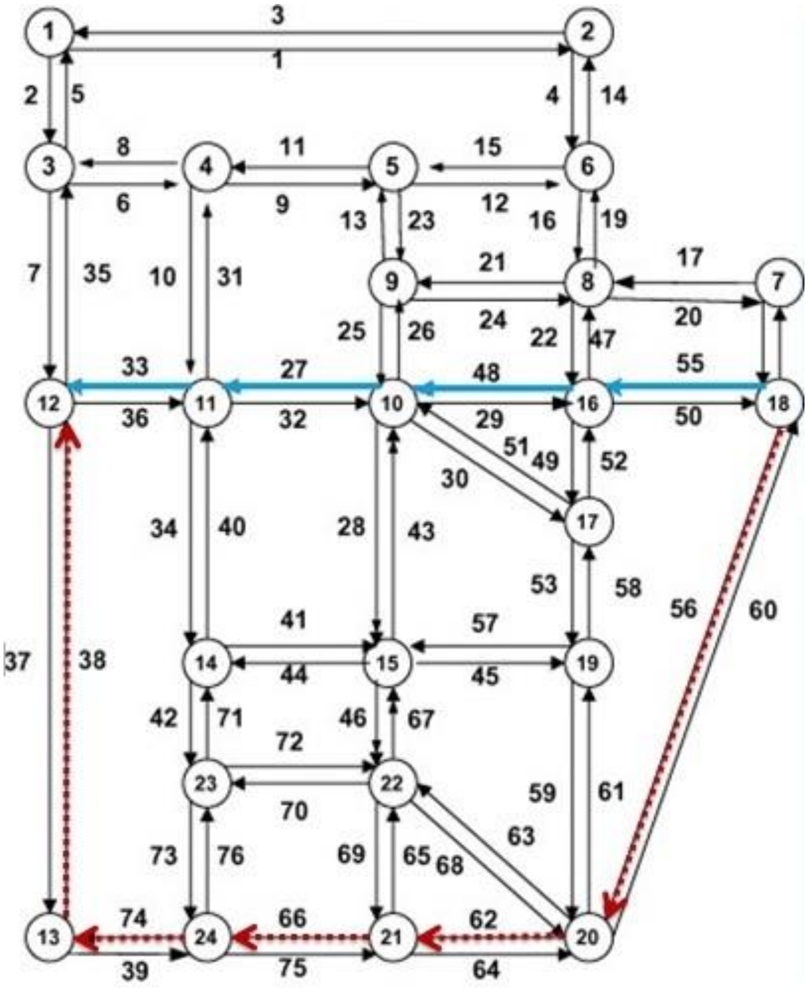


Figure 1. Sioux Falls test network with node and link numbers

As a motivating example, to illustrate the importance for solving Eqn (1), even in case of relatively small error probabilities, a well-known, medium sized Sioux Falls Network shall be used, as shown in Figure 1. The link costs data associated with the links are assumed to be given by the free flow travel times reported in Suwansirikul et al. [24], multiplied by 100. Without loss of generality, suppose that one is interested in traveling from node 18 to node 12. The optimal route in the classical shortest path framework would be 18-16-10-11-12, with an (expected) travel time of 19 minutes (cf. solid bold arrows in Figure 1).

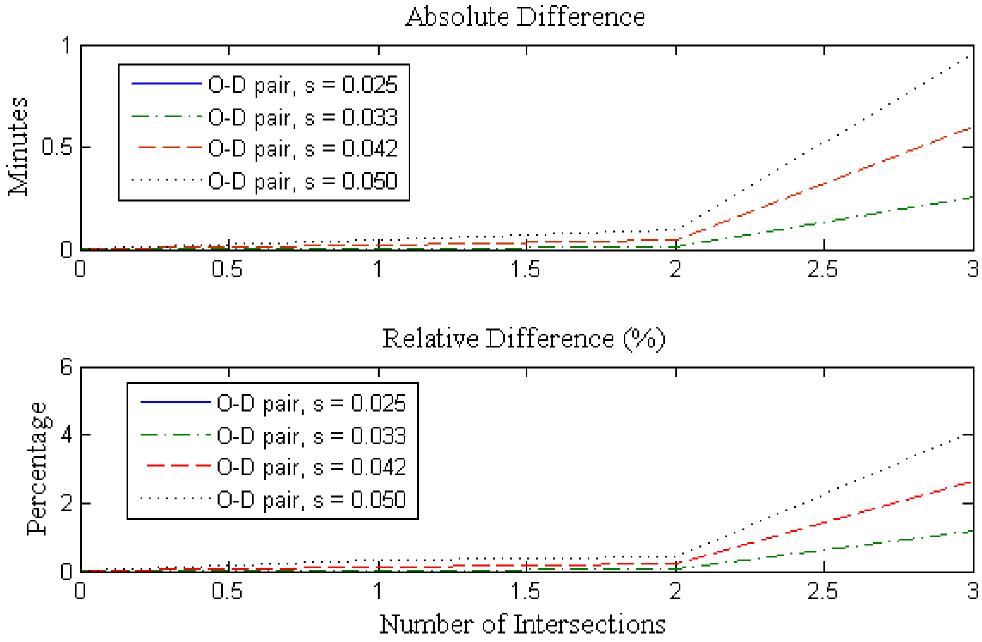


Figure 2. Travel time difference between proactive and reactive routing strategies for O-D pair (18 - 16) of Sioux Falls network

Let us set

$$p_i = (|U(i)| - 1)s \qquad (2)$$

Here, s is used to scale the probability. For the first experiment, s is set to 0.025 which makes p_{16} = 0.075, p_{10} = 0.10, and p_{11} = 0.075. The optimal route, derived from the model, remains as same as the classical shortest path with these slight chances of mistakes. In the next experiment, s is set to 0.033 (That is, p_{16} = 0.10, p_{10} = 0.13, and

p_{11} = 0.10). The chances of mistakes, associated with three intermediate nodes, are not high in this experiment. But the result shows (refer to Figure 2), following the route 18-20-21-24-13-12, prescribed by the model, optimizes the travel time by 1.17%. The prescribed route has less complex intersections with less chance of making mistakes on the intermediate nodes (p_{20} = 0.10, p_{21} = 0.067, p_{24} = 0.067, p_{13} = 0.033). The average chance of making mistake per intermediate node on the classical shortest route is 1.64 times the average chance of mistakes on the route prescribed by the model. In the next two experiments, s is increased to 0.042 and 0.050 successively to demonstrate the efficiency of the model in terms of saving travel time. For $s = 0.050$, the highest error probability, associated with intermediate node 20 is 20% which is relatively small. Even with such small chance of making mistakes, not only optimal route chances, but also travel time can be saved by nearly 1 minute. This saving in travel time might seem modest, however, one should bear in mind that a relatively small network has been used. In the next section, large-scale, real-world transportation networks will be examined to more thoroughly investigate this issue.

2.2 MODEL'S IMPLICATION IN REAL WORLD

While accounting for the possibility of missing turns is theoretically appealing in terms of modeling realism, in practice, it is not clear what the expected benefits are in terms of travel time savings. To this end, the proactive routing strategy resulting from Eqn (1) is compared next that explicitly anticipates the possibility of mistakes with the following commonly used reactive strategy: Determine a shortest route from a user-specified origin to the desired destination. Every time the driver misses a turn/ deviates from the route, a new shortest route will be determined based on the new location the driver ends up at. Since the expected travel times are minimized in Eqn (1), it is clear that the above described reactive strategy will always lead to a higher expected travel time. However, it is not clear how large this difference will be. In the remainder of this section the Anaheim, CA and Austin, TX road networks will be used to demonstrate that the differences can be substantial. Intuitively, this difference should be a function of the number of intersections between an O-D pair, and the error probabilities.

In the first numerical experiment, the effect of the number of intersections between a given O-D pair on the difference between the expected travel time from a proactive and reactive routing strategy is investigated. In the first experiment, $s = 0.033$ is selected in Eqn (2). Since the Anaheim network has at most 6 outgoing links for a node, there is a 16.67% chance of making mistakes on that link. (On average, the Anaheim network has 2.2 outgoing links per node. Hence the error probability is, on average, 0.04, a rather small value.). One O-D pair (295 - 5) is then selected at random (the shortest travel time between this O-D pair is 16.88 minutes). For this O-D pair, the expected travel times under both routing strategies are then calculated, and starting from each node along the classical shortest path. Backtracking along this path, starting from the destination node, a path with increasingly more intersections is then built. Figure 3 shows the results of this exercise. As can be seen from the upper part in the figure, the absolute difference between the two routing strategies indeed, as expected, increase (or remains constant) when the number of intersections increase. For the O-D pair (295 -5), the classical shortest path recommends to take the route where there is 1 intermediate node (node 337) with 6 outgoing links, 7 intermediate nodes with 5 outgoing links (node 308, node 361, node 378, node 394, node 407, node 406, node 402), and 3 intermediate nodes (node 405, node 404, node 401) with 4 outgoing links. The driver needs to cross those 11 complex (of total 26 intersections) intersections with higher probability of making mistakes to reach the destination according to classical shortest path. When the error probabilities are associated with driving, the travel time on classical shortest route becomes 21.73 minutes. The solution algorithm recommends an alternate path where those 11 complex intersections were avoided. Although a route with more intersections (33 intersections) is prescribed, there is no intersection having more than 3 outgoing links. The travel time for the O-D pair of the solution algorithm is 17.97 minutes. That is, the algorithm is able to recommend an alternate route with optimized travel time and less complex intersections. For the same O-D pair (295-5), the scaling factor is reduced to 0.025 and 0.017. (This corresponds to an error probability of about 0.050 and 0.036 successively on average on the route prescribed by classical shortest path algorithm). The results are also shown in Figure 3. As can be seen, although the difference becomes smaller, it remains significant. In other words, these results suggest that it is of critical importance solve (1), instead of a classical shortest path problem when human error is possible.

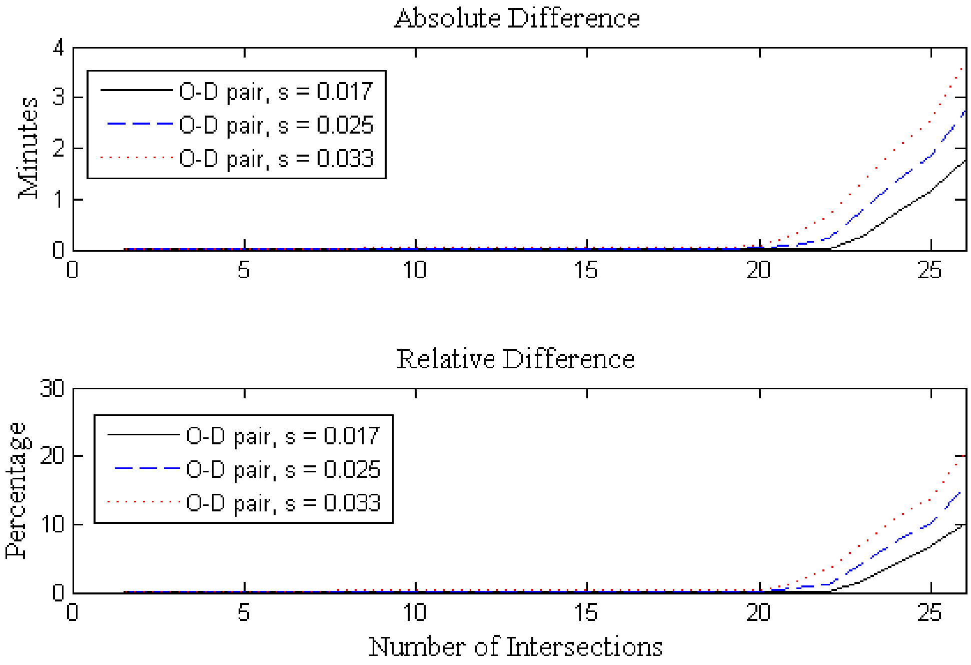


Figure 3. Travel time difference between proactive and reactive routing strategies for O-D pair (295 - 5) of Anaheim network

In the second numerical example, it is shown how the number of complex intersections affect travel time even when the travel time between an O-D pair is not that significant. In this example, an O-D pair (209-25) of Anaheim network is chosen randomly. The shortest travel time between the O-D pair is 9.68 minutes with 20 intersections in between. The interesting incidence about this O-D pair is that, there are 7 intersections which have more than 4 outgoing links in their classical shortest route (node 394, node 378, node 361, node 337, node 308, node 273, and node 269). The solution algorithm recommends an alternate route with 21 intersections avoiding 6 of these complex intersections. The absolute difference and relative difference between two routing strategies are shown in Figure 4, with different scale factors. When s is set to 0.033, we have $p_i \leq 0.13$ on this shortest route (on average, the probability of mistakes is 0.08), which is rather a small value. But when error in driving is accounted for, the absolute difference can be more than 2 minutes even if the route is comparatively smaller. Even the relative difference between the routing strategies can be more than 20% (shown in lower part of Figure 4). In the next two experiments, the scaling factor is reduced to 0.025 and 0.017. As can be seen from the Figure 4, even with a very small scale factor 0.017 (This corresponds to an average error probability of about 0.039 for the intersections of the O-D pair), the relative difference between two routing strategies can be more than 10%.

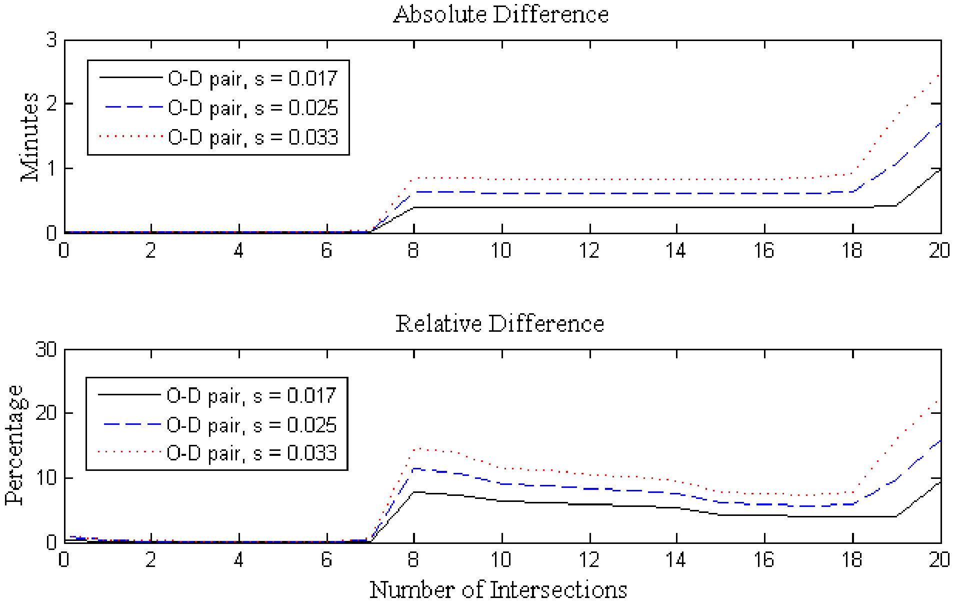


Figure 4. Travel time difference between proactive and reactive routing strategies for O-D pair (209 - 25) of Anaheim network

In the next example, an O-D pair (1-400) of Anaheim network is chosen which has 33 intersections in its classical shortest route. The route, prescribed by classical shortest path algorithm, does not have any intersection more than three outgoing links (there are 1.47 outgoing links on average on the classical shortest route which results in the error probabilities of 0.015 on average for the intermediate nodes). The effect of the absence of complex intersections is investigated on the shortest route with three different scale factors (0.033, 0.025, and 0.017). It can be seen from Figure 5 that with the highest scaling factor 0.033 (which results in the error probabilities 0.016 on average on the classical shortest route), however, the maximum absolute difference between the routing strategies does not exceed half a minute. Even the maximum relative difference is less than 1% (shown in lower part of Figure 5). Although the algorithm recommends an alternate optimized route with more intersections (34 intersections), the differences between the routing strategies are not significant because of the absence of complex intersections on the shortest route.

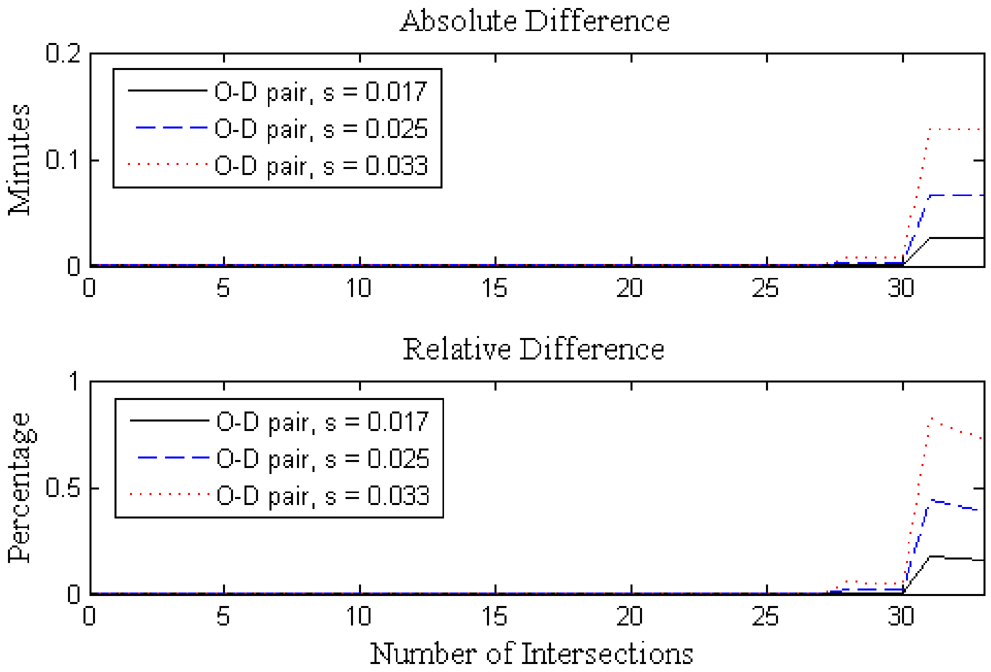


Figure 5. Travel time difference between proactive and reactive routing strategies for O-D pair (1 - 400) of Anaheim network

At this point, for demonstrating the importance of solving (1) rather than solving classical shortest path, a large scale network named Austin, TX (7388 nodes, 18961 links) shall be used.

An O-D pair (278-6772) of Austin network was chosen randomly which has 107 intermediate nodes in its classical shortest route. In the first experiment the scaling factor was set to be 0.033 (this sets the average probability of 0.07 for the nodes of the classical shortest route). It is to be noted that this O-D pair does not have any intermediate nodes more than 4 outgoing links (average number of outgoing links per intermediate node is 3.09). Having 4 outgoing links (including the outgoing link back to its predecessor node which corresponds to a U turn in for an intersection) for an intersection is a very common kind of intersection as can be seen in real life. The classical shortest travel time between the O-D pair is 128.90 time unit (since it is not mentioned in BarGera [25] whether the time unit for Austin network is minute or hour). When mistakes in driving are considered with the scale factor $s = 0.033$, the travel time on the shortest route becomes 156.62 unit. The developed solution algorithm recommends an alternate route with 159 intermediate nodes which is able to optimize the travel time to 144.97 unit. The difference between the routing strategies goes up to nearly 12 unit (shown in upper part of Figure 6) for some nodes to reach the destination node. This example demonstrates that the cumulative effect of the error is significant with the increasing number of intersections between an O-D pair even when the intersections are not that complex.

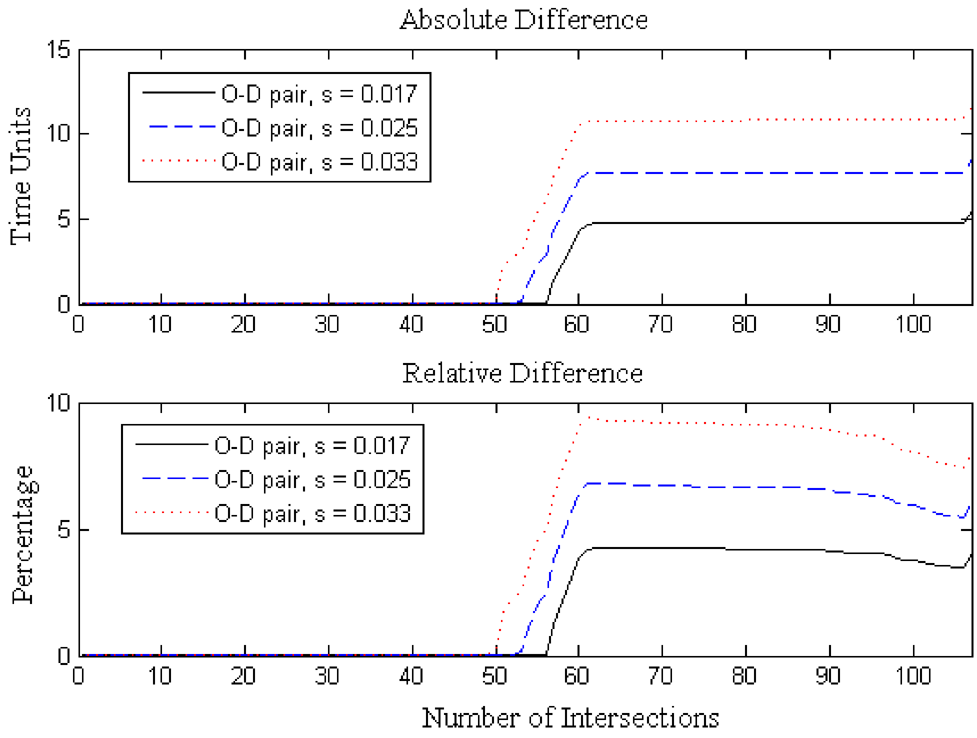


Figure 6. Travel time difference between proactive and reactive routing strategies for O-D pair (278-6772) of Austin network

Another O-D pair (449-6772) of Austin network has been chosen randomly to show the importance of solving (1) rather than solving the classical shortest path algorithm. This O-D pair has 85 intersections, in its classical shortest route. In the first experiment, the scaling factor *s* is set to be 0.033. Once the path is built after backtracking from the destination node to the source node, it is clearly seen from Figure 7, that the model is able to minimize around 10 time unit for nearly 20 intermediate nodes to reach the destination node 6772. The solution algorithm has more than twice as many numbers of intersections as the classical shortest algorithm has. Despite the fact that the solution algorithm recommends an alternate route with 172 intersections, it is able to minimize 5 time unit expected travel time for some intermediate nodes of the O-D pair (449-6772) at the lowest scale factor $s = 0.017$ of this exercise.

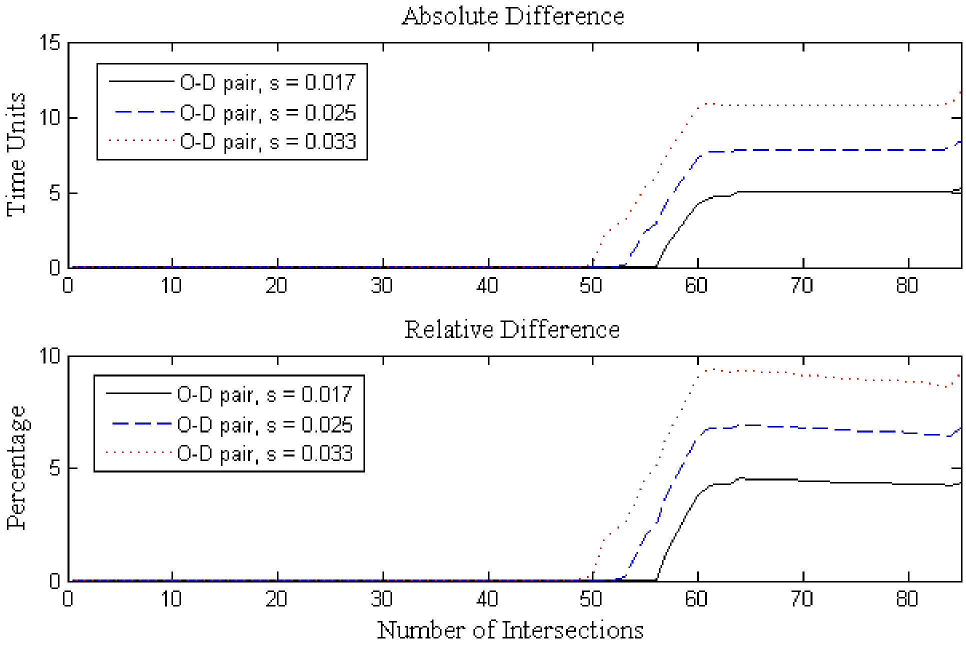


Figure 7. Travel time difference between proactive and reactive routing strategies for O-D pair (449-6772) of Austin network

The case study, as shown in Figure 8, an O-D pair (263-6199) of Austin network with 227 intermediate nodes is chosen. The simulation was run with three different values of scaling factor. Surprisingly, the expected travel time from each intersection to the destination is not minimized significantly, although there is relatively large number of intersections between the O-D pair in the classical shortest route. More insight was investigated to find the reason behind such incidence. There is no intersection in the shortest route which has more than 4 outgoing links. Even the intersections having 4 outgoing links is very few in number on the route. Moreover the average number of outgoing links per node is 2.16. Even with the highest scaling factor ($s = 0.033$) used for the experiment, we have 3.87% chance of making mistakes on average per intersections.

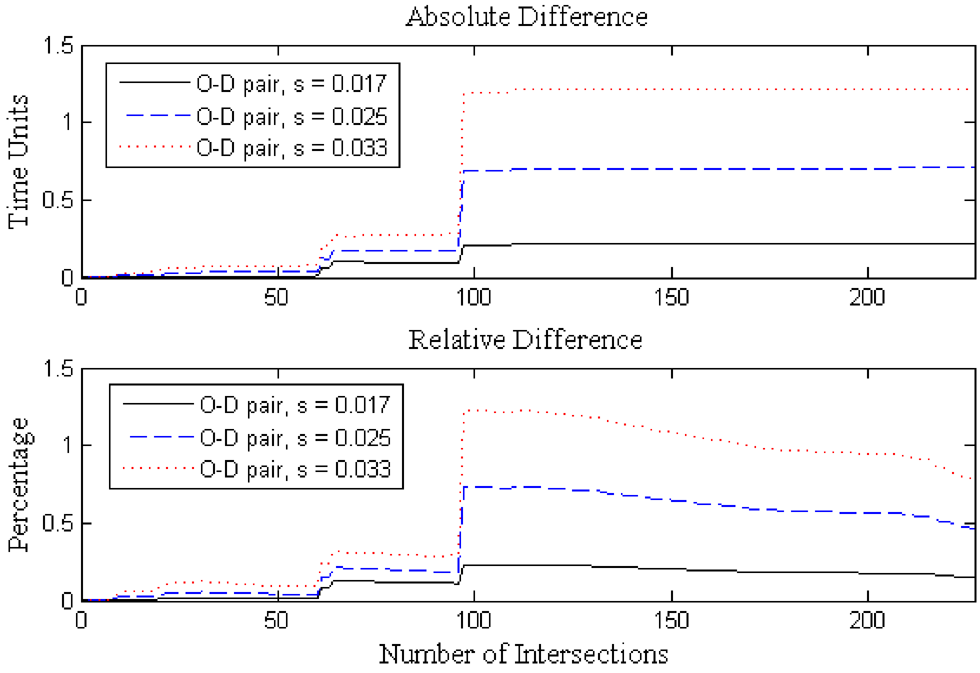


Figure 8. Travel time difference between proactive and reactive routing strategies for O-D pair (263-6199) of Austin network

The numerical example, shown in Figure 9, demonstrates how the change in probability of mistakes affects the difference between two routing strategies. In the first two experiments the scaling factor s was set to be 0.017. Both absolute difference and relative difference do not become significant. In the next experiment, the scaling factor was increased to 0.025, which sets 5.25% chance of making mistakes on average per node while following the shortest route. With this small chance of making mistakes, the model can not minimize the expected travel time for most of the intermediate nodes on the shortest route to reach the destination node. When setting the scaling factor s to be 0.033, as can be seen from both upper and lower part of Figure 9, there are sudden significant changes in the difference of expected travel time for some intermediate nodes under both routing strategies. This result is quite different from the experiment conducted for the O-D pair (263-6199), as shown in Figure 8. Although the O-D pair (6766-1100) has relatively small number of intersections compared to the O-D pair (263-6199), the average number of links per nodes on the classical shortest route between O-D par (6766-1100) is 3.10, a rather larger value than the average number of outgoing links (average is 2.16) per intermediate nodes between the O-D pair (263-6199).

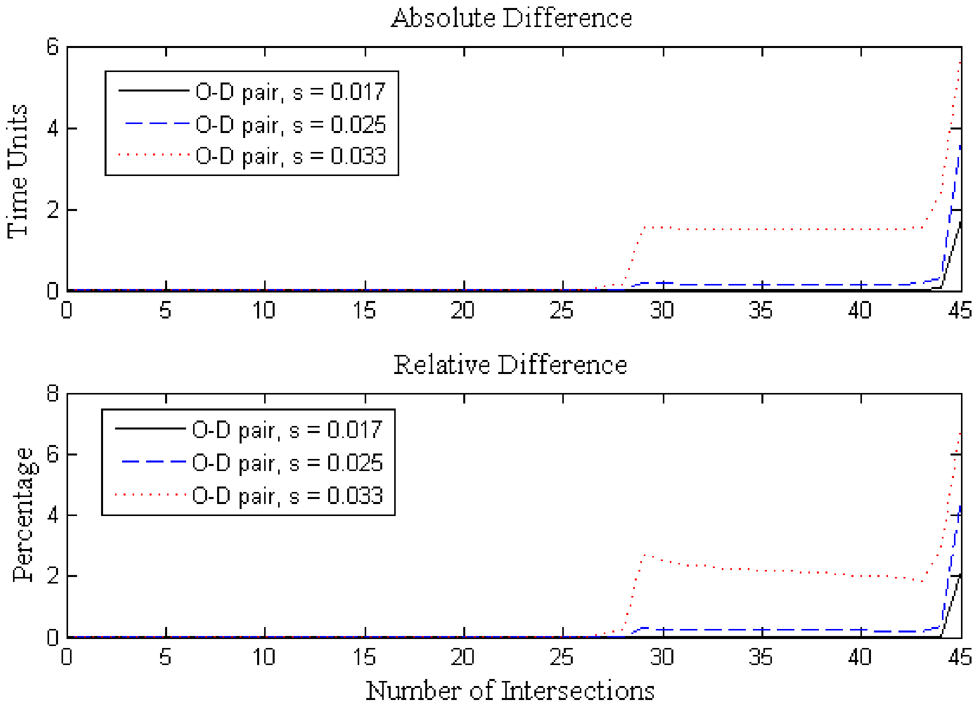


Figure 9. Travel time difference between proactive and reactive routing strategies for O-D pair (6766-1100) of Austin network

CHAPTER 3

SOLUTION APPROACH

There are lots of traditional algorithms which are capable of minimizing the expected travel time by selecting the best action possible at each node. The goal (of a navigation system) is to choose a set of control strategies $u \in U(i)$ at each node i (taking into account the possibility that the directions provided might not be followed), so that the destination is reached within the shortest possible time (in expected value It should be noted that truly large scale MDPs demand efficient implementation of solution algorithm. Hence a well-known solution approach, policy iteration algorithm (e.g. see Puterman [2]), has been adapted to improve its efficiency. Moreover the value iteration [4] algorithm was implemented because of its space efficiency and also for validating the result found from policy iteration. They are described step by step in this chapter along with the convergence criteria we have used for experiments.

3.1 POLICY ITERATION ALGORITHM

Policy iteration (PI) is an iterative procedure in the space of deterministic policies where it discovers the optimal policy by generating a sequence of monotonically improving policies. One of the main contributions of the thesis is the development of an enhancement which we make to standard policy iteration. Hence, the basic policy iteration is briefly reviewed in this section. The steps in policy iteration can be summarized as follows:

- **Step 1 (Initialization)**: Initialize by choosing an initial policy $u(i)$ for all nodes $i \neq D$. In this research, we initialize by finding the classical shortest path for each node to the destination node. The algorithm for finding initial policy is briefly discussed below:

Algorithm:

```
for each node $i \neq D$ to the destination node $D$
    find shortest path from $i$ to $D$ following basic label correcting
    algorithm.
    select the successor node $f$ of node $i$ to reach destination node $D$ as the
    chosen action to take from node $i$. That is, $u(i) \leftarrow f$.
end for
```

To illustrate this step, consider Figure 10 where the numbers on the links denote the associate travel time (in minutes). The shortest path from node 1 to node 6 is 1-2-5-6. After following the algorithm mentioned above, $u(1)$ = 2. Similarly, $u(2)$ = 5, $u(3)$ = 2, $u(4)$ = 6, $u(5)$ = 6. A vector of control strategies $\bar{u}_l$ is introduced to store the control strategies $u \in U(i)$ for all nodes $i \neq D$. Here l denotes the number of iteration. For initialization step, l = 0.

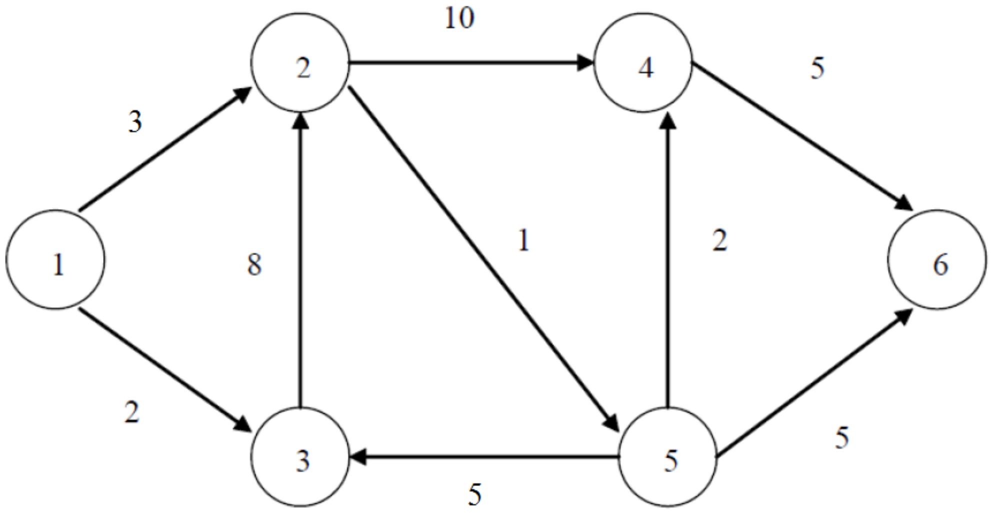


Figure 10. A network with node numbers and link costs

From the Bellman's equation, mentioned in Eqn (1), it is clear that the solution to this set of equations depends directly on the choice of action in each state. Since the chosen states from all states from the initial control strategies are known, the chances of mistakes for all nodes are put back to Eqn (1) in order to obtain the labels $e(i)$. That is the following system of linear equations needs to be solved:

$$e(1) = (1 - p_1)\big(3 + e(2)\big) + p_1(2 + e(3))$$

$$or,\ e(1) - (1 - p_1)e(2) - p_1 e(3) = 1 - p_1 \qquad (3)$$

$$e(2) = p_2\big(10 + e(4)\big) + (1 - p_2)\big(1 + e(5)\big)$$

$$or, e(2) - (1 - p_2)e(5) - p_2 e(4) = 1 + 9p_2 \qquad (4)$$

$$e(3) = 8 + e(2)$$

$$or, e(3) - e(2) = 8 \tag{5}$$

$$e(4) = 5 \tag{6}$$

$$e(5) = \frac{p_5}{2}\big(5 + e(3)\big) + \frac{p_5}{2}\big(2 + e(4)\big) + (1 - p_5)\ 5$$

$$or,\ \ e(5) - \tfrac{p_5}{2}e(3) - \tfrac{p_5}{2}e(4) = \ 5 - \tfrac{3p_5}{2} \tag{7}$$

The above system can be formulated as $\big(I - P(\bar{u}_0)\big).e^{\bar{u}_0} = C$, where $\big(I - P(\bar{u}_0)\big)$ is the co-efficient matrix of the system, I is the identity matrix, and C is a vector with elements $C(i, \bar{u}_0)$ $.e^{\bar{u}_0}$ is the vector of expected travel times from each node to the destination node when the classical shortest path has been chosen as the control strategy.

$$I - P(\bar{u}_0) = \begin{bmatrix} 1 & -(1-p_1) & -p_1 & 0 & 0 \\ 0 & 1 & 0 & -p_2 & -(1-p_2) \\ 0 & -1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & -\frac{p_5}{2} & -\frac{p_5}{2} & 1 \end{bmatrix}, \text{ and}$$

$$C = \begin{pmatrix} 1 - p_1 \\ 1 + 9p_2 \\ 8 \\ 5 \\ 5 - \frac{3p_5}{2} \end{pmatrix}$$

- **Step 2 (Policy Evaluation)**: Solve the system of linear equation $\big(I - P(\bar{u}_l)\big).e^{\bar{u}_l} = C$ where $.e^{\bar{u}_l}$ is the vector of expected travel times from each node to the destination node, when the vector of control strategies $\bar{u}_l$ has been in use at l^{th} iteration. I is the identity matrix, and C is a vector with elements $C(i, \bar{u}_l)$. To simplify notation, in the following chapters K will be used to denote the matrix $I - P(\bar{u}_l)$.

- **Step 3 (Policy Improvement)**: Solve Bellman's equation (cf. Puterman [2]; Bertsekas [3]) to obtain the next set of control strategy u , i.e.:

$$e^{\bar{u}_l}(i) = \min_{u \in U(i)}[C(i, \bar{u}_l) + \sum p_{ij}(\bar{u}_l) \,.\, e^{\bar{u}_l}(j)\,] \qquad (8)$$

  This set of control strategy is at least as good as the previous one for getting optimal solution, if not better.

- **Step 4 (Convergence Test)**: Repeat the above steps from Step 2 until the difference in $e^{\bar{u}_l}$ in subsequent iterations is smaller than a prescribed error tolerance. The stopping criteria $|e^{\bar{u}_l}(i) - e^{\bar{u}_{l-1}}(i)| < \in$ will be used, where $\in$ denotes the error tolerance.

The algorithm, described above, can also be referred to as actor-critic architectures [26]. Figure 11 shows a block diagram of policy iteration (or an actor-critic architecture) and the dependencies among the various components. Policy improvement is also known as the actor and policy evaluation is known as the critic, because the actor is responsible for the way the model acts following the given policy and the critic is responsible for criticizing the way the model acts.

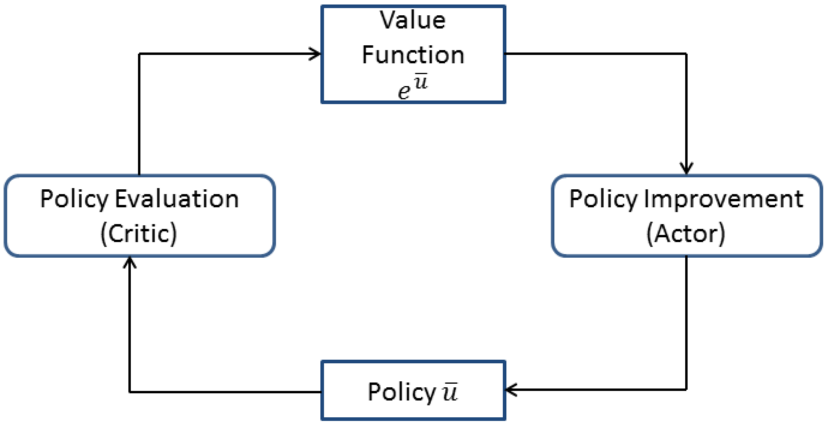


Figure 11. Policy Iteration (Actor-Critic architecture)

3.2 VALUE ITERATION ALGORITHM

Value Iteration (VI) is an algorithm which successively approximates the value function, starting from an arbitrary initial estimate. It is a robust and well-known algorithm for solving an MDP, but not considered viable because of its slow convergence nature for large scale problems. Although throughout the thesis, this algorithm is not the subject for enhancement, it is reviewed because of its space efficiency for some networks. Also, the optimal solution of this approach was used to validate the results found after solving both normal and enhanced policy iteration. The steps for the algorithm are given below:

- **Step 1**: Initialize by guessing an initial expected travel time $e(i)$ for all nodes $i \neq D$.
- **Step 2**: Solve Bellman's equation (cf. Puterman [2]; Bertsekas [3]) to obtain the next set of minimized expected travel time $e(i)$ for all nodes $i \neq D$, i.e.:

$$e(i) = \min_{u \in U(i)} [\, C(i,u) + \textstyle\sum_j p_{ij}(u) e(j) \,] \tag{9}$$

- **Step 3:** Repeat the above step until the difference in $e(i) = 1,2, \ldots.., n$ in subsequent iterations is smaller than a prescribed error tolerance (∈). ∈ denotes the error tolerance for finding the optimized travel time for each node to the destination node.

The algorithm iterates over every state, and updates the value of that state according to Equation 3.2. Value iteration can therefore be viewed as generating all one-step optimal policies, then two-step optimal policies, etc., and is generally considered a form of dynamic programming which opts to approximate the expected travel time from each source node to the destination node within some error tolerance.

CHAPTER 4

STORAGE SCHEME

This work is about the efficient storage scheme that has been used while developing the solution algorithm and conducting numerical experiments with real world large scale data. This chapter discusses how the large scale transportation data are stored into sparse matrix storage framework.

4.1 SPARSE MATRIX REPRESENTATION FOR TRANSPORTATION NETWORK

In this thesis, the popular and highly efficient “sparse matrix” storage scheme, used by the “sparse equations solver” research community for handling large-scale transportation network applications was adapted. In general, the travel time for each link is stored in a n x n node-node adjacency matrix. But this kind of storage is prohibitively expensive. As for example, each row of such matrix of Austin, TX network contains around 3 non-zero values, on average, only among all 7,388 elements. An efficient sparse matrix storage scheme for large scale transportation network was adapted from Lawson et al. [27]. For a more detailed discussion, the reader is referred to Nguyen [5].

In actual computer implementation, the sparse matrix representation of node-node adjacency matrix demands more in order to find optimal routes efficiently from a network where all links are not bidirectional. In addition to the sparse storage scheme of Nguyen [5], a new sparse storage scheme for storing incoming link information is introduced. The network partitioning step of Domain Decomposition algorithm needs to know which nodes are connected to a node when it is being explored in order to be a member of a subdomain. This step considers all neighboring nodes with any nature of the connectivity (e.g. outgoing or incoming). To our knowledge, most of the real world networks found from Bar-Gera [25] are not bidirectional. This additional storage scheme, used mainly in Domain Decomposition algorithm, helps to keep the basic sparse matrix representation [5] unchanged all throughout the developed solution algorithm. To illustrate the sparse matrix representation scheme, a network which contains 5 nodes and 8 links is used, see Figure 12.

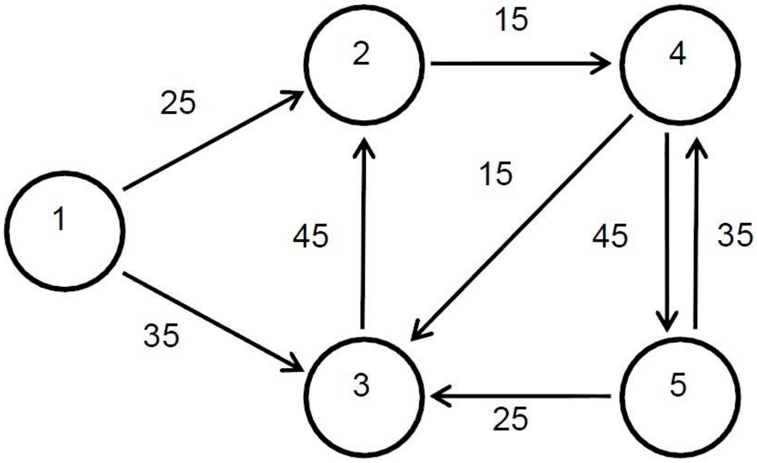


Figure 12. A simple 5 node/ 8 link network [27]

The node-node incidence type matrix of this network is the 5x5 matrix A.

$$A = \begin{bmatrix} 0 & 25 & 35 & 0 & 0 \\ 0 & 0 & 0 & 15 & 0 \\ 0 & 45 & 0 & 0 & 0 \\ 0 & 0 & 15 & 0 & 45 \\ 0 & 0 & 25 & 35 & 0 \end{bmatrix}$$

This matrix A can be stored in a sparse format using user-defined arrays *NZ, IA*, *JA* and C. Arrays *IA* and *JA* are used to index outgoing nodes associated with the link cost values stored in A. By following the sparse matrix representation scheme of Lawson et al. [27], for the network shown in Figure 12 we have

$$NZ = \begin{pmatrix} 2 \\ 1 \\ 1 \\ 2 \\ 2 \end{pmatrix}, JA = \begin{pmatrix} 2 \\ 3 \\ 4 \\ 2 \\ 3 \\ 5 \\ 3 \\ 4 \end{pmatrix}, IA = \begin{pmatrix} 1 \\ 1+2 \\ 3+1 \\ 4+1 \\ 5+2 \\ 7+2 \end{pmatrix} = \begin{pmatrix} 1 \\ 3 \\ 4 \\ 5 \\ 7 \\ 9 \end{pmatrix}, C = \begin{pmatrix} 25 \\ 35 \\ 15 \\ 45 \\ 15 \\ 45 \\ 25 \\ 35 \end{pmatrix}$$

Three new user-defined vectors *NZ_I, XA*, *KA* were introduced for indexing incoming nodes associated with the same link cost values of *C*. For instance, for the above network, we have:

$$NZ\_I = \begin{pmatrix} 0 \\ 2 \\ 3 \\ 2 \\ 1 \end{pmatrix}$$

For example, *NZ_I*(1)=0 tells us that there are no incoming links to node 1, and so on. The array can then be easily computed by recursively adding the vector *NZ* to the vector *XA*. By definition, to initialize, the first entry in *XA* is set equal to 1. (The length of *XA* is the number of nodes plus 1.) For instance, for the above network, we have:

$$XA = \begin{pmatrix} 1 \\ 1+0 \\ 1+2 \\ 3+3 \\ 6+2 \\ 8+1 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 3 \\ 6 \\ 8 \\ 9 \end{pmatrix}$$

That is, in order to calculate *XA*(i), we have added *NZ_I*(i-*1*) to *XA*(i-*1*). Based on *XA* we can, for example, easily recover the number of non-zeros in column 5 of A by the substraction *XA*(6)- *XA*(5) = 1, and so on. In conjunction with *XA*, one also needs to define the row number of the non-zero terms in *C*. In the example, we have for *KA*:

$$KA = \begin{pmatrix} 1 \\ 3 \\ 1 \\ 4 \\ 5 \\ 2 \\ 5 \\ 4 \end{pmatrix}$$

That is, there is no incoming link for node 1 as *XA(2)*- *XA(1)*= 0 (no non zero values in the first column of *A*). *KA*(1)=1: the first link comes to node 2 from node 1, and *KA*(2)=3: the first link comes to node 2 from node 3 (note that from *XA* one knows that the number of non-zeros in the second column of *A* is equal to 2; hence, one knows that the first two entries in *KA* corresponds to node 2 and so on). The final task is to store the link cost in vector C_I, with same dimension as *KA.*

$$C\_I = \begin{pmatrix} 25 \\ 45 \\ 35 \\ 15 \\ 25 \\ 15 \\ 35 \\ 45 \end{pmatrix}$$

Thus instead of using a n x n node-node adjacency square matrix, the whole network can be represented using these *NZ* (n x 1), *IA* ($n + 1$ x 1), *JA* (m x 1), C(m x 1), *NZ_I* (n x 1), *XA* ($n + 1$ x 1), *KA* (m x 1), and C_I(m x 1)vectors only.

4.2 STORING MDP MODEL

In this section, a discussion on how the co-efficient matrix is filled up for one iteration only is provided. Hence the subscript l is dropped, which denotes the iteration count index, for better readability. In the *policy evaluation* step of policy iteration algorithm, a system of linear equations $\left(I - P(\bar{u})\right).e^{\bar{u}} = C$ is solved. This is too expensive in terms memory to store I and $P(\bar{u})$ since both of the matrix will be of nxn dimension. Rather, a framework is proposed to calculate the co-efficient matrix, resulting from $\left(I - P(\bar{u})\right)$. The implicit assumption for making mistakes in driving, made by Ng and Sathasivan [1], is that all wrong turns are equally likely. Based on this assumption, a vector $\bar{p}$ of dimension $nx1$ is enough to store p_i for all nodes $i \neq D$. This vector stores the error probabilities associated with each node. The co-efficient matrix K, for each iteration l, is filled up according to the model described below. Note that $K_{i,i} = 1$ for all nodes $i \neq D$.

$$K_{i,j|j\in U(i)} = \begin{cases} -(1-p_i), & \mathit{if\ node\ j\ is\ chosen\ according\ to\ current\ policy} \\ -\dfrac{p_i}{|U(i)|-1}, & \mathit{else} \end{cases}$$

CHAPTER 5

ENHANCED POLICY ITERATION

In this chapter, a novel solution algorithm is presented to solve the MDP model described in Chapter 2. The Domain Decomposition (DD) algorithm of Johnson et al. [22, 23] is used to decompose the transportation network in such a way that a system of linear equations (i.e., in Step 2 of the policy iteration algorithm) can be reduced to several smaller, decoupled systems of equations. A simplified variation of the domain decomposition scheme is employed. The reader is encouraged to refer to Johnson et al. [22, 23] for the most recent research into the subject. The chapter is concluded after presenting how this thesis make use of the DD algorithm to solve a system of linear equation with respect to a given policy.

5.1 DOMAIN DECOMPOSITION SCHEME

In this section, the domain decomposition scheme is briefly reviewed. To facilitate this discussion, the partitioning algorithm will be applied to the example problem given by Figure 13. Further, the assumption is made to partition the network into three sub-domains. It is important to note that the actual link cost is not needed, and for this algorithm the cost for every link is one. Lastly, as mentioned in Chapter 4, the sparse matrix storage scheme is able to handle any kind of link directionality, it does not matter if the network partitioned, is bi-directional or uni-directional. The effort is to identify nodes which are adjacent to, or alternatively, far from other nodes.

The major objective in all domain decomposition algorithms is to minimize the total number of system boundary nodes, and to have sub-domains of approximately equal size (with the former having more priority than the latter). The first requirement ensures that the communication time between sub-domains will be minimized, while the second criterion ensures an (approximately) equal workload for each sub-domain. The steps involved during the domain partitioning phase can be summarized as follows:

Step 1: The transportation network topology (shown in Figure 13) can be represented by the matrix notation shown in Figure 14. In actual computer implementation, this matrix will be efficiently stored in the popular sparse storage scheme [5].

Step 2: Rank (*R*) or Degree of a node is a number which represents how many neighboring nodes a node has. R can be found by the connectivity information stored in *IA, JA, XA, and KA. R* is found by the UNION set operation on the number of occurrences a given node found as both incoming node and outgoing node. For example, in Figure 12, the set of outgoing nodes from node 4 is {3, 5}, and the set of nodes from where the links come to node 4 is {2, 5}. The UNION set operation is used to get {2, 3, 5} as the neighbors of node 4. Thus *R* = 3 for node 4.

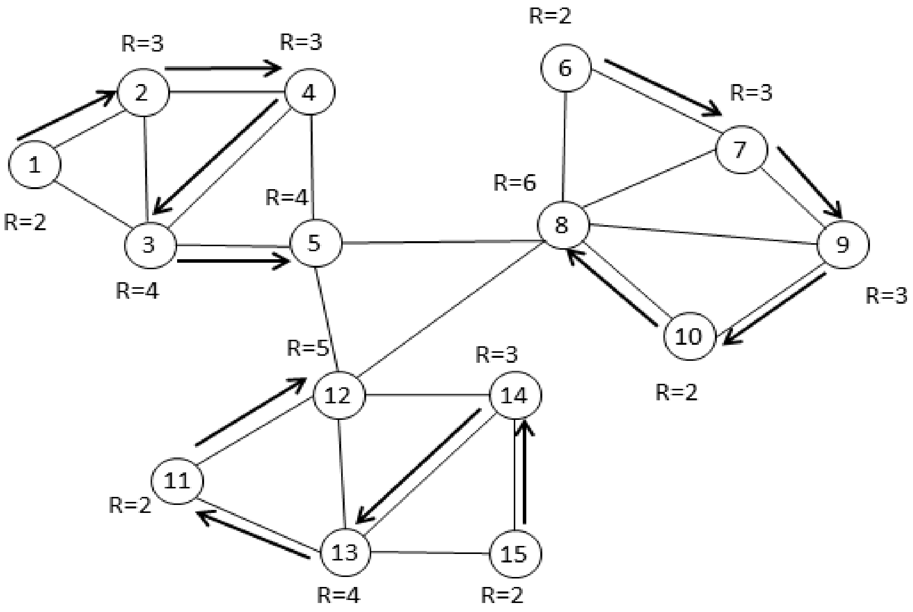
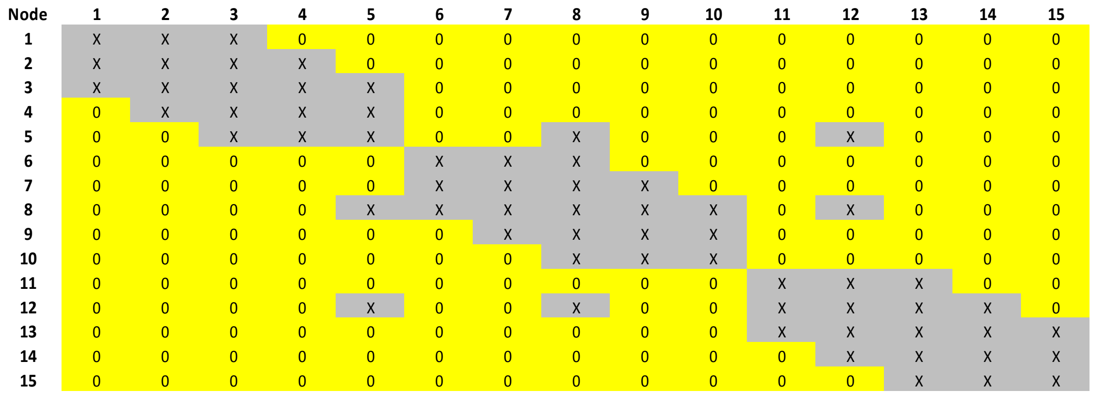


Figure 13. A network with nodal rank and directions for populating sub-domains



Node	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	X	X	X	0	0	0	0	0	0	0	0	0	0	0	0
2	X	X	X	X	0	0	0	0	0	0	0	0	0	0	0
3	X	X	X	X	X	0	0	0	0	0	0	0	0	0	0
4	0	X	X	X	X	0	0	0	0	0	0	0	0	0	0
5	0	0	X	X	X	0	0	X	0	0	0	X	0	0	0
6	0	0	0	0	0	X	X	X	0	0	0	0	0	0	0
7	0	0	0	0	0	X	X	X	X	0	0	0	0	0	0
8	0	0	0	0	X	X	X	X	X	X	0	X	0	0	0
9	0	0	0	0	0	0	X	X	X	X	0	0	0	0	0
10	0	0	0	0	0	0	0	X	X	X	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	X	X	X	0	0
12	0	0	0	0	X	0	0	X	0	0	X	X	X	X	0
13	0	0	0	0	0	0	0	0	0	0	X	X	X	X	X
14	0	0	0	0	0	0	0	0	0	0	0	X	X	X	X
15	0	0	0	0	0	0	0	0	0	0	0	0	X	X	X

Figure 14. Node-node adjacency matrix for network representation of Figure 13

Step 3: Upon examination of Figure 13, it is seen there are 5 nodes (1, 6, 10, 11, 15) which have the same (lowest) rank; one arbitrarily selects the smallest node number (node 1) of these low ranking nodes as the starting (interior) node for sub-domain 1. Performing the Label Correcting Algorithm (LCA) with the source node as node 1 (to find the shortest path from the source node to all other nodes in the system) will reveal that node 15 is "very far" from node 1. Hence, node 15 is used as the starting (interior) node for the second sub-domain. Performing the LCA with the source node 15 will reveal that nodes 6, 9 and 10 are "far away" from both nodes 1 and 15. Because nodes 6 and 10 have a lower rank than node 9 they will be considered for the final starting (interior) node. Node 6 is arbitrarily selected (note that if node 10 were selected, the results would remain unchanged) as the starting (interior) node for the third (last) sub-domain (refer to Figure 13).

Step 4: Each sub-domain will be "sequentially" populated from the remaining nodes, based on the simple heuristic rule: the next j-th node to be added to a k-th sub-domain will be the node which has the lowest rank which shares a direct connection between either the source node or any other node in the populated sub-domain. Figure 13 shows the order in which the sub-domains are populated. (It should be noted that a more efficient partitioning can be achieved, in larger examples, if one were to modify the rule above by first searching for the node which is closest to the source node and then, as a tiebreaker, examine the rank of the node as described above).

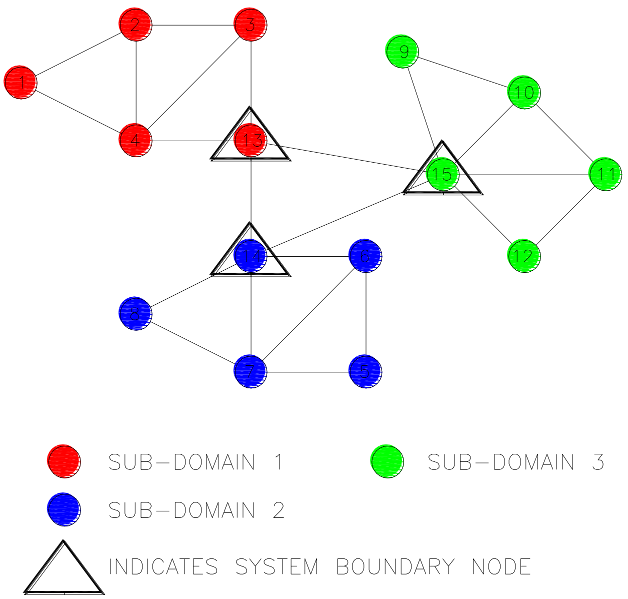
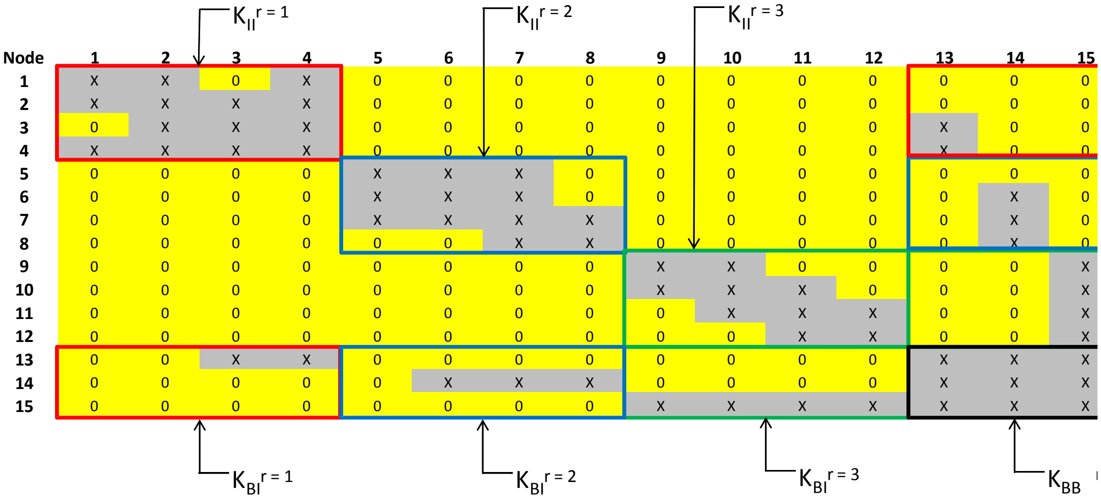


Figure 15. The network, shown in Figure 13 after renumbering

Step 5: System boundary nodes (and sub-domains' interior nodes) can be identified by considering each link of the network based on the following simple rule: If the nodes which define a link belong to the same sub-domain, then these two nodes are considered interior nodes, however, if these nodes belong to two different sub-domains, then both nodes must be considered system boundary nodes. The determination of system boundary nodes is imperative for the next step.

Step 6: Finally, the nodes in the system were renumbered. Starting with sub-domain 1, The nodes which belong to each sub-domain were numbered sequentially, however, ignoring the boundary nodes. Once all of the interior nodes have been renumbered, the boundary nodes were numbered as the last nodes in the set as shown as by Figures 15 and 16.


$K_{II}{}^{r=1}$
$K_{II}{}^{r=2}$
$K_{II}{}^{r=3}$
$K_{BI}{}^{r=1}$
$K_{BI}{}^{r=2}$
$K_{BI}{}^{r=3}$
K_{BB}

Node	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	X	X	0	X	0	0	0	0	0	0	0	0	0	0	0
2	X	X	X	X	0	0	0	0	0	0	0	0	0	0	0
3	0	X	X	X	0	0	0	0	0	0	0	0	X	0	0
4	X	X	X	X	0	0	0	0	0	0	0	0	X	0	0
5	0	0	0	0	X	X	X	0	0	0	0	0	0	0	0
6	0	0	0	0	X	X	X	0	0	0	0	0	0	X	0
7	0	0	0	0	X	X	X	X	0	0	0	0	0	X	0
8	0	0	0	0	0	0	X	X	0	0	0	0	0	X	0
9	0	0	0	0	0	0	0	0	X	X	0	0	0	0	X
10	0	0	0	0	0	0	0	0	X	X	X	0	0	0	X
11	0	0	0	0	0	0	0	0	0	X	X	X	0	0	X
12	0	0	0	0	0	0	0	0	0	0	X	X	0	0	X
13	0	0	X	X	0	0	0	0	0	0	0	0	X	X	X
14	0	0	0	0	0	X	X	X	0	0	0	0	X	X	X
15	0	0	0	0	0	0	0	0	X	X	X	X	X	X	X

Figure 16. Node-node adjacency matrix after reordering the matrix shown in Figure 14

Now that the re-numbering of the nodes has been performed, a new node-node adjacency matrix can be created (for Figure 14). This is shown in Figure 5.4. As can be seen, after reordering the model (per Step 6 of the algorithm outlined above), described in Section 2, the adjacency matrix has been decomposed effectively into sub-matrices K_{II}^{r}, K_{IB}^{r}, K_{BI}^{r}, K_{BB}. K_{II}^{r} simply represents the adjacency matrix of the smaller, sub-structured systems. Sub-matrices K_{IB}^{r}, K_{BI}^{r}, and K_{BB} represents the coupling effects among sub-domains. The matrix in Figure 16 gets decoupled after discarding rows and column 13, 14, and 15 (as node 13, 14, and 15 are the boundary nodes for the network in Figure 15). For a more detailed discussion, the authors refer the reader to Nguyen [5]. Vector e and C will be reordered according to the new numbering of the nodes; e will be found as $e_I^r(r = 1, 2, 3 \ldots NS$, where NS denotes the number of sub domains) and e_B and C will be found as C_I^r and C_B.

5.2 ENHANCED POLICY ITERATION

This section describes how the system of linear equation $K.e = C$ is solved. It is described in Step 2 of Section 3.1 with the help of sub-matrices K_{II}^{r}, K_{IB}^{r}, K_{BI}^{r}, and K_{BB}. It should be noted that this system is solved with respect to a policy at one iteration only. To increase readability the superscript $\bar{u}_l$ will be dropped, as mentioned in Chapter 3, 0all throughout this chapter, which denotes the policy (or control strategy) to get the optimal solution at each iteration of the model. The derivation is given below:

$$\begin{pmatrix} {K_{II}}^{1} & \cdots & \cdots & \cdots & {K_{IB}}^{1} \\ \vdots & {K_{II}}^{2} & \cdots & \cdots & {K_{IB}}^{1} \\ \vdots & \vdots & \ddots & \cdots & \vdots \\ \vdots & \vdots & \vdots & {K_{II}}^{NS} & {K_{IB}}^{NS} \\ {K_{BI}}^{1} & {K_{BI}}^{2} & \cdots & {K_{BI}}^{NS} & K_{BB} \end{pmatrix} \begin{pmatrix} {e_I}^{1} \\ {e_I}^{2} \\ \vdots \\ {e_I}^{NS} \\ e_B \end{pmatrix} = \begin{pmatrix} {C_I}^{1} \\ {C_I}^{2} \\ \vdots \\ {C_I}^{NS} \\ C_B \end{pmatrix}$$

The above matrix notation, except for the last row, can be broken down into a system of linear equations, as shown below:

$${K_{II}}^{1}.{e_I}^{1} + {K_{IB}}^{1}.e_B = {C_I}^{1}$$

$${K_{II}}^{2}.{e_I}^{2} + {K_{IB}}^{2}.e_B = {C_I}^{2}$$

$$\vdots$$

$${K_{II}}^{NS}.{e_I}^{NS} + {K_{IB}}^{NS}.e_B = {C_I}^{NS}$$

The set of equations, mentioned above, can be re-written as following with $r = 1, 2, \dots, NS$:

$${K_{II}}^{r}.{e_I}^{r} + {K_{IB}}^{r}.e_B = {C_I}^{r} \tag{10}$$

From the last row of the matrix , the following linear equation is obtained:

$${K_{BI}}^{1}.{e_I}^{1} + {K_{BI}}^{2}.{e_I}^{2} + \cdots + {K_{BI}}^{NS}.{e_I}^{NS} + K_{BB}.e_B = C_B$$

$$\sum_{r=1}^{NS}({K_{BI}}^{r}.{e_I}^{r}) + K_{BB}.e_B = C_B \tag{11}$$

It is seen that the above matrix notation is broken down into two linear equations, as shown in Eqn (10) and Eqn (11).

From Eqn (10), e_I can be solved as follows:

$$e_I{}^r = \ ({K_{II}}^r)^{-1}({C_I}^r - {K_{IB}}^r.e_B) \tag{12}$$

Substituting Eqn (12) into Eqn (11) we obtain:

$$\textstyle\sum_{r=1}^{NS}({K_{BI}}^r.({K_{II}}^r)^{-1}({C_I}^r - {K_{IB}}^r.e_B)) + K_{BB}.e_B = C_B$$

The above equation can be re-written as:

$$(K_{BB} - \textstyle\sum_{r=1}^{NS} {K_{BI}}^r({K_{II}}^r)^{-1}{K_{IB}}^r).e_B = C_B - \textstyle\sum_{r=1}^{NS} {K_{BI}}^r.({K_{II}}^r)^{-1}.{C_I}^r \tag{13}$$

Eqn (13) can now be simplified as:

$$\overline{K}_{BB}.e_B = \ \bar{C}_{BB} \tag{14}$$

where:

$$\overline{K}_{BB} = \ K_{BB} - \textstyle\sum_{r=1}^{NS} {K_{BI}}^r({K_{II}}^r)^{-1}{K_{IB}}^r \tag{15}$$

$$\bar{C}_{BB} = C_B - \textstyle\sum_{r=1}^{NS} {K_{BI}}^r.({K_{II}}^r)^{-1}.{C_I}^r \tag{16}$$

Eqn (15) and Eqn (16) are substituted into Eqn (14) and solve for e_B. Now e_B can be inserted back into Eqn (12).

Thus e_I^r (for r = 1, 2, … , NS) and e_B are obtained as our solution. For a more detailed discussion about the computation from the decomposed sub-matrices, the reader is referred to Nguyen [5]. To this end, a direct solution method is used for solving the large scale system. This is the method for solving a large linear system with the exact solution e within a finite number of steps, provided that all arithmetic operations are exact. Direct solution methods are attractive because of their generality, reliability, and efficiency. The choice of an appropriate solution method depends on the main goal, which is to run the application as fast as possible. Moreover, running the application at all given computer resources (disk space, and memory space) is another important concern. In this section, a very basic form of direct solution approach will be discussed. Note that the linear system, to be solved, is unsymmetrical most of the cases. This limits us in using some well-known direct methods for solving the system. Since the co-efficient matrix, formulated from transportation network, is sparse, a built in function of Matlab named "sparse()" has been invoked on each of the sub-matrices K_{II}^r, K_{IB}^r, K_{BI}^r, and K_{BB} before operations.

- **Step 1**: Calculate $\overline{K}_B$ from the equation $\overline{K}_B = \sum_{r=1}^{NS}(K_{BI}^r \,.\, ({K_{II}}^r)^{-1}. K_{IB}^r)$.
- **Step 2**: Using the matrix $\overline{K}_B$, calculate $\overline{K}_{BB}$ from the equation $\overline{K}_{BB} = K_{BB} - \overline{K}_B$.
- **Step 3**: Calculate $\bar{C}_B$ from the equation $\bar{C}_B$=$\sum_{r=1}^{NS}(K_{BI}^r \,.\, ({K_{II}}^r)^{-1}. C_I^r)$.
- **Step 4**: Using the vector $\bar{C}_B$, calculate $\bar{C}_{BB}$ from the equation $\bar{C}_{BB} = C_B - \bar{C}_B$.
- **Step 5**: Solve system of linear equations $\overline{K}_{BB}. e_B = \bar{C}_{BB}$
- **Step 6**: Calculate e_I^r from the formula e_I^r =$({K_{II}}^r)^{-1}$. (C_I^r - K_{IB}^r.e_B). for $r = 1, 2, 3, \ldots . NS$.

It is important to retain the mapping between the previous node numbering and new node numbering, such that the optimized travel time from each source node to the destination node can be obtained for the original problem.

CHAPTER 6

PERFORMANCE COMPARISON AND DISCUSSION

This chapter deals mainly with the comparison of solution time needed to obtain the optimized travel time following different solution approaches. To validate the research conducted in this thesis, 5 networks (Anaheim, Winnipeg, Chicago Sketch, Austin, Chicago Regional) are used from Bar-Gera [25]. The code which generated all the results, shown in this chapter, was written in MATLAB. The performance data were found using a computer with an Intel(R) Core(TM) 2 Quad CPU Q9300 @ 2.50 GHz processor and 8 GB of RAM.

For each of the networks, some destination nodes were randomly selected for getting the expected travel time needed to reach those destination nodes from all source nodes using different solution approaches. The solving time and iteration counts were recorded for each of the solution approaches for each destination node separately without applying DD partitioning algorithm. At first, the performance of value iteration (VI) with policy iteration (PI) is compared. Then in the next table, the developed "enhanced policy iteration" that helps to speed up the solution time is shown. Direct solution method includes a basic linear system solving approach under the column "PI". Note that all matrices and sub-matrices are used after invoking built-in "sparse()" before operations. The direct solver employs LAPACK with optimized BLAS, developed by Greg Henry of the University of Tennessee and Intel Corp. (Moler, 2000), for solving system of linear equation. It should also be noted that the workspace, used for all simulation runs has 4 processors. Although the results are shown for upto 8 smaller subdomains, the solving time, shown in the tables, has used maximum of 4 processors.

Table 1 shows the results of three simulation runs (with three randomly chosen destination nodes) of Anaheim network (416 nodes). For this network, policy iteration shows better results than value iteration in terms of both iteration count and solving time.

Table 1. Performance comparison between VI and PI without applying DD algorithm (Anaheim network)

Destination Node	VI		PI	
	Iteration Count	Solving Time (seconds)	Iteration Count	Solving Time (Seconds)
5	78	0.4457	4	0.2318
50	69	0.4294	3	0.1759
400	80	0.4922	5	0.2967
280	88	0.5267	4	0.2680

In Table 2, the comparison of solving time of enhanced policy iteration with different solution approaches is shown after using the sub-matrices K_{II}^{r}, K_{IB}^{r}, K_{BI}^{r}, and K_{BB} following DD partitioning algorithm, in order to get the solution with the same destinations. The simulations were run using single processor and multicore processors. As can be seen from the table, DD partitioning algorithm minimizes the solving time for some simulation runs while using single processor. The parallel implementation does not speed up the solution time for Anaheim network because the solving time at each iteration for this network is relatively small. The communication time among the processors adds extra noise to the performance for parallel implementation of the developed solution approach.

Table 2. Performance comparison of PI after applying DD partitioning algorithm (Anaheim network)

Destination Node	# Sub-Domains	PI (Seconds)	
		Serial	Parallel
5	2	0.2361	0.5835
	3	0.1818	0.5740
	4	0.1652	0.6196
	5	0.1478	0.6512
	6	0.1496	0.6005
	7	0.1369	0.6321
	8	0.1666	0.6283
50	2	0.1697	0.4842
	3	0.1386	0.4838
	4	0.1133	0.5325
	5	0.1134	0.5904
	6	0.1143	0.5533
	7	0.0978	0.5358
	8	0.1136	0.5594
400	2	0.2843	0.7203
	3	0.2338	0.7021
	4	0.1734	0.7172
	5	0.1634	0.7541
	6	0.1602	0.7545
	7	0.1645	0.7403
	8	0.1729	0.7952
280	2	0.2338	0.5729
	3	0.1856	0.5972
	4	0.1425	0.6267
	5	0.1351	0.6674
	6	0.1400	0.6753
	7	0.1398	0.6471
	8	0.1527	0.8503

Next, all the solution approaches were tested with two medium size networks, collected from Bar-Gera [25], named Winnipeg (1,052 nodes) and Chicago Sketch (933 nodes). The results, obtained from several simulation runs, agree with the results obtained by testing Anaheim network. The results are shown in Table 3, Table 4, Table 5, and Table 6 respectively. As can be seen, parallel implementation using multicore processors yields worse performance than using a single processor for the solution approach. For the parallel implementation, Matlab's built-in "Parallel Computing Toolbox" has been used as a black-box. Such implementation could not tell us more about the reasons behind such slow performance. Moreover, operations on small and medium sized matrices happen so fast that the computational time is hard to measure accurately. Hence, all the solution approaches were tested with two large scale network, Austin (7,388 nodes) and Chicago Regional (12,982 nodes).

Table 3. Performance comparison between VI and PI without applying DD algorithm (Winnipeg network)

Destination Node	VI		PI	
	Iteration Count	Solving Time (seconds)	Iteration Count	Solving Time (Seconds)
35	492	7.1010	4	1.3116
250	550	8.7738	3	1.0501
1000	333	4.8067	5	1.6312
550	673	7.6547	5	1.6395

Table 4. Performance comparison among different solution approaches of PI after applying DD partitioning algorithm (Winnipeg network)

Destination Node	# Sub-Domains	PI (Seconds)	
		Serial	Parallel
35	2	1.6606	1.8303
	3	1.1234	1.3121
	4	0.8604	1.2220
	5	0.7679	1.2138
	6	0.8193	1.3425
	7	0.7570	1.3443
	8	0.7150	1.3326
250	2	1.2715	1.3742
	3	0.8511	1.0218
	4	0.6439	0.9724
	5	0.5737	0.9582
	6	0.5807	1.0124
	7	0.5650	1.0334
	8	0.5359	0.9831
1000	2	2.0941	2.1618
	3	1.3775	1.6379
	4	1.0408	1.5570
	5	0.9855	1.5630
	6	0.9707	1.5963
	7	0.9508	1.5676
	8	0.9116	1.6587
550	2	2.0749	2.6094
	3	1.4183	2.2596
	4	1.0632	1.9574
	5	0.9573	1.9902
	6	0.9543	2.0603
	7	0.9441	2.0935
	8	0.8982	2.1202

Table 5. Performance comparison between VI and PI without applying DD algorithm (Chicago Sketch network)

Destination Node	VI		PI	
	Iteration Count	Solving Time (seconds)	Iteration Count	Solving Time (Seconds)
333	3189	44.2296	4	1.0778
654	2904	39.3855	3	0.9761
831	3189	42.8940	4	1.0666
920	5809	48.1724	3	0.8947

Table 6. Performance comparison among different solution approaches of PI after applying DD partitioning algorithm (Chicago Sketch network)

Destination Node	# Sub-Domains	PI (Seconds)	
		Serial	Parallel
333	2	1.0637	1.3237
	3	0.7801	1.0914
	4	0.7121	1.1519
	5	0.6513	1.1889
	6	0.5844	1.1334
	7	0.5328	1.1743
	8	0.7276	1.6493
654	2	0.7904	1.0645
	3	0.5651	0.8322
	4	0.5279	1.0358
	5	0.5038	0.9531
	6	0.4479	0.8768
	7	0.4142	1.0385
	8	0.5597	1.0523
831	2	1.0637	1.3237
	3	0.7801	1.0914
	4	0.7121	1.1519
	5	0.6513	1.1889
	6	0.5844	1.1334
	7	0.5328	1.1743
	8	0.7276	1.6493
920	2	0.7896	1.0418
	3	0.5703	0.8678
	4	0.5175	0.8954
	5	0.5050	0.9559
	6	0.4517	0.9898
	7	0.3917	0.8978
	8	0.5436	1.0810

It is surprising to observe that for Austin and Chicago Regional networks (see Table 7 and Table 9 respectively), VI algorithm solves the problem faster than PI algorithm. The larger the size of a network is, the slower the solving time gets for policy iteration compared to value iteration. Partitioning the domain into several numbers of subdomains helps the PI running faster in serial processor implementation although it does not outperform the value iteration algorithm. The solving time is getting closer with the increase in the number of subdomain.

Table 7. Performance comparison between VI and PI without applying DD algorithm (Austin network)

Destination Node	VI		PI	
	Iteration Count	Solving Time (seconds)	Iteration Count	Solving Time (Seconds)
5	2013	105.13	27	453.65
2345	1681	90.92	14	236.65
4200	2021	106.41	20	333.68
6432	2212	118.46	13	238.91

Unlike the three networks (Anaheim, Winnipeg, and Chicago Sketch), as shown in Table 8 and Table 10, the parallel implementations of the solution approach with 4 processors catch the solving time of basic PI (without DD partitioning algorithm), even sometimes perform better than the single processor implementation. This result is inspiring in the sense that more processors will be able to solve the model with less amount of time if the network is partitioned efficiently into more number of subdomains. The less the number of boundary nodes, the more efficient the partitioning algorithm is. In that case a smaller system needs to be solved as shown in Eqn (14). Moreover the architecture of the workspace, used for all simulation runs, might cause the slower performance in communication time for parallel implementation. After recording the computational time associated with each subdomain separately, it is clear that parallel implementation adds huge noise as the communication time between processors in such computations and thus results in surprisingly slow performance.

Table 8. Performance comparison among different solution approaches of PI after applying DD algorithm (Austin network)

Destination Node	# Sub-Domains	PI (Seconds)	
		Serial	Parallel
5	2	555.96	448.27
	3	359.02	295.75
	4	299.89	262.47
	5	245.15	224.42
	6	194.87	193.07
	7	196.73	193.63
	8	196.31	210.08
2345	2	289.72	236.33
	3	186.54	160.47
	4	158.28	165.87
	5	127.21	135.87
	6	100.14	96.20
	7	101.91	118.97
	8	100.94	114.73
4200	2	425.57	343.15
	3	274.80	233.11
	4	233.32	208.15
	5	182.05	178.93
	6	144.22	138.58
	7	152.67	170.91
	8	149.66	164.46
6432	2	271.05	238.74
	3	175.77	141.09
	4	146.44	139.58
	5	119.62	109.49
	6	96.94	109.39
	7	96.84	95.26
	8	94.34	97.42

Table 9. Performance comparison between VI and PI without applying DD algorithm (Chicago Regional network)

Destination Node	VI		PI	
	Iteration Count	Solving Time (seconds)	Iteration Count	Solving Time (Seconds)
1111	1433	318.63	11	871.33
8568	3705	340.13	10	964.18
400	1915	185.34	8	733.34
12150	1917	178.81	8	801.81

Table 10. Performance comparison among different solution approaches of PI after applying DD algorithm (Chicago Regional network)

Destination Node	# Sub-Domains	PI (Seconds)	
		Serial	Parallel
1111	2	798.05	721.79
	3	670.32	689.34
	4	582.41	739.92
	5	475.76	689.53
	6	427.21	666.89
	7	488.79	611.19
	8	438.48	570.28
8568	2	848.19	732.74
	3	573.44	526.23
	4	579.69	581.25
	5	499.14	537.06
	6	474.98	526.98
	7	436.31	516.18
	8	430.91	504.70
400	2	643.99	613.06
	3	455.67	420.77
	4	450.56	448.27
	5	387.60	411.00
	6	368.73	420.06
	7	336.09	409.92
	8	331.38	396.17
12150	2	725.06	604.14
	3	452.90	420.01
	4	446.09	464.25
	5	380.80	421.94
	6	372.01	409.23
	7	332.44	395.87
	8	325.03	407.87

CHAPTER 7

CONCLUSION

Algorithms (i.e., shortest path algorithm) in navigation systems assume that drivers are able to follow the directions without mistakes. However, this is not always the case, especially when the drivers are passing through complex intersections and the graphical interface design of navigation systems is poor. Using real-world transportation network, it has been shown that this is indeed the case and that the “classical” shortest routes do not remain optimal anymore

The focus of the research conducted in the thesis is to develop a robust, reliable, and efficient algorithm to solve “driving with error” models. Through real-world transportation network examples (such as Anaheim, Austin, Winnipeg, Chicago Sketch, and Chicago Regional), it has been demonstrated that the proposed solving algorithm, coupled with DD partitioning, significantly reduces the computational time (as compared to the traditional algorithm, which operates directly on the entire large network) for the basic policy iteration algorithm.

Various aspects of the mentioned problem have still been left unexplored, and hence, can be considered in future works. For example, it was intended to devise discrete choice models to estimate the error probability as a function of intersection and personal characteristics of the driver. It was also intended to develop models that are robust to the misspecification of the error likelihoods.

A more principled approach to network partitioning is necessary to ensure the minimization of system boundary node count which is responsible for adding overhead to the computation cost. Partitioning the network into as many subdomains as possible, following a more efficient partitioning algorithm [22, 23], and distributing the subdomains to the processors of a supercomputer (with 32 or 64 processors) for solving the model could show more exciting results.

Moreover, the parallel implementation tasks were left totally to Matlab's "Parallel Computing Toolbox". Such implementation should be done carefully and efficiently after investigating which portions of the code segments take more time while using multicore processors.

It was also observed that for a transportation network, the average number of links per node is almost same regardless the size of the network. That is, the bigger networks have more sparse system than the smaller ones compared to the size of network. A multi-frontal method (i.e., Davis [28], Davis [29], Davis, [30]), or a hybrid multi-frontal method (i.e., Amestoy et al. [31], Amestoy et al. [32], Xia et al. [33], Raju et al. [34]) coupled with DD partitioning algorithm can be developed to take the advantage of such sparsity for better performance. UMFPACK library [28], used by "Backlash" operator, takes way more advantage of a sparse system by using multi-frontal method. Since the source code for this library is open for all, it can be embedded in the DD framework for taking more advantage of the sparse sub-matrices obtained from DD partitioning algorithm.

In this thesis, a direct solution method coupled with DD framework was implemented. An efficient iterative solution method, a hybrid of BI-CG and BI-CGSTAB (Van der Vorst [35]) algorithm, named IBI-CGSTAB (Yang and Brent [36]), could be employed which reduces the global communication cost of the parallel performance significantly. Moreover, constructing appropriate preconditioned matrix using parallel and multilevel methods, discussed in Saad and Van Der Vorst [37], for obtaining super-linear convergence for iterative solvers, could be an interesting future research direction.

REFERENCES

[1] M. W. Ng and K. Sathasivan, "Probabilistic Modeling of Erroneous Human Response to In-Vehicle Route Guidance Systems: A First Look," *Journal of Intelligent Transportation Systems,* 2013.

[2] M. L. Puterman, *Markov decision processes: discrete stochastic dynamic programming* vol. 414: Wiley.com, 2009.

[3] D. P. Bertsekas, *Dynamic programming and optimal control* vol. 1: Athena Scientific Belmont, 1995.

[4] R. Bellman, *Dynamic Programming*: Princeton University Press, 1957.

[5] D. T. Nguyen, *Finite Element Methods: Parallel-Sparse Statics and Eigen-Solutions*: Springer, 2006.

[6] M. Ng, J. Park, and S. T. Waller, "A hybrid bilevel model for the optimal shelter assignment in emergency evacuations," *Computer‐Aided Civil and Infrastructure Engineering,* vol. 25, pp. 547-556, 2010.

[7] M. Ng and S. T. Waller, "Reliable evacuation planning via demand inflation and supply deflation," *Transportation Research Part E: Logistics and Transportation Review,* vol. 46, pp. 1086-1094, 2010.

[8] Y. Yin and H. Yang, "Simultaneous determination of the equilibrium market penetration and compliance rate of advanced traveler information systems," *Transportation Research Part A: Policy and Practice,* vol. 37, pp. 165-181, 2003.

[9] W. Szeto and H. K. LO, "The impact of advanced traveler information services on travel time and schedule delay costs," in *Intelligent Transportation Systems*, 2005, pp. 47-55.

[10] T. Toledo and R. Beinhaker, "Evaluation of the potential benefits of advanced traveler information systems," *Journal of Intelligent Transportation Systems,* vol. 10, pp. 173-183, 2006.

[11] R. L. Bertini and D. J. Lovell, "Impacts of sensor spacing on accurate freeway travel time estimation for traveler information," *Journal of Intelligent Transportation Systems,* vol. 13, pp. 97-110, 2009.

[12] A. Higatani, T. Kitazawa, J. Tanabe, Y. Suga, R. Sekhar, and Y. Asakura, "Empirical analysis of travel time reliability measures in Hanshin expressway network," *Journal of Intelligent Transportation Systems,* vol. 13, pp. 28-38, 2009.

[13] N. Uno, F. Kurauchi, H. Tamura, and Y. Iida, "Using bus probe data for analysis of travel time variability," *Journal of Intelligent Transportation Systems,* vol. 13, pp. 2-15, 2009.

[14] H. Rakha, I. El-Shawarby, and M. Arafeh, "Trip travel-time reliability: issues and proposed solutions," *Journal of Intelligent Transportation Systems,* vol. 14, pp. 232-250, 2010.

[15] A. Chen, H. Yang, H. K. Lo, and W. H. Tang, "Capacity reliability of a road network: an assessment methodology and numerical results," *Transportation Research Part B: Methodological,* vol. 36, pp. 225-252, 2002.

[16] H. K. Lo and Y.-K. Tung, "Network with degradable links: capacity analysis and design," *Transportation Research Part B: Methodological,* vol. 37, pp. 345-363, 2003.

[17] H. K. Lo, X. Luo, and B. W. Siu, "Degradable transport network: travel time budget of travelers with heterogeneous risk aversion," *Transportation Research Part B: Methodological,* vol. 40, pp. 792-806, 2006.

[18] A. Sumalee, D. P. Watling, and S. Nakayama, "Reliable network design problem: case with uncertain demand and total travel time reliability," *Transportation Research Record: Journal of the Transportation Research Board,* vol. 1964, pp. 81-90, 2006.

[19] M. Ng and S. T. Waller, "A computationally efficient methodology to characterize travel time reliability using the fast Fourier transform," *Transportation Research Part B: Methodological,* vol. 44, pp. 1202-1219, 2010.

[20] M. Ng, W. Szeto, and S. Travis Waller, "Distribution-free travel time reliability assessment with probability inequalities," *Transportation Research Part B: Methodological,* vol. 45, pp. 852-866, 2011.

[21] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, "Network flows: theory, algorithms, and applications," 1993..

[22] P. Johnson, D.T. Nguyen, and M. Ng, “A New Shortest Distance Domain Partitioning Algorithm for Large-Scale Transportation Network Problems”, Structures Research Technical Note No. 07-26-2013, CEE Department, 135 KAUF, Old Dominion University, Norfolk, VA 23529, 2013.

[23] P. Johnson, D.T. Nguyen, and M. Ng, “A New Shortest Distance Domain Partitioning Algorithm for Large-Scale Transportation Network Problems”, submitted to the TRB’2014 Conference, Washington, D.C, 2014.

[24] C. Suwansirikul, T. L. Friesz, and R. L. Tobin, "Equilibrium decomposed optimization: a heuristic for the continuous equilibrium network design problem," *Transportation Science,* vol. 21, pp. 254-263, 1987.

[25] H. Bar-Gera, "Transportation network test problems," ed, 2012. Available: http://www.bgu.ac.il/~bargera/tntp/.

[26] A. G. Barto, R. S. Sutton, and C. W. Anderson, "Neuronlike adaptive elements that can solve difficult learning control problems," *Systems, Man and Cybernetics, IEEE Transactions on,* pp. 834-846, 1983.

[27] G. Lawson, S. Allen, G. Rose, D. Nguyen, and M. Ng, "Parallel Label Correcting Algorithms for Large-Scale Static and Dynamic Transportation Networks On Laptop Personal Computers," in *Proceedings of the Transportation Research Board's 92nd Annual Meeting*, 2013.

[28] T. A. Davis, "Umfpack version 5.6.2 user guide," *Department of Computer and Information Science and Engineering, University of Florida, Gainesville, FL,* 2013.

[29] T. A. Davis, "Algorithm 832: UMFPACK V4. 3---an unsymmetric-pattern multifrontal method," *ACM Transactions on Mathematical Software (TOMS),* vol. 30, pp. 196-199, 2004.

[30] T. A. Davis, "A column pre-ordering strategy for the unsymmetric-pattern multifrontal method," *ACM Transactions on Mathematical Software (TOMS),* vol. 30, pp. 165-195, 2004.

[31] P. R. Amestoy, I. S. Duff, J.-Y. L'Excellent, and J. Koster, "A fully asynchronous multifrontal solver using distributed dynamic scheduling," *SIAM Journal on Matrix Analysis and Applications*, vol. 23, pp. 15-41, 2001.

[32] P. R. Amestoy, A. Guermouche, J.-Y. L’Excellent, and S. Pralet, "Hybrid scheduling for the parallel solution of linear systems," *Parallel computing,* vol. 32, pp. 136-156, 2006.

[33] J. Xia, S. Chandrasekaran, M. Gu, and X. S. Li, "Superfast multifrontal method for large structured linear systems of equations," *SIAM Journal on Matrix Analysis and Applications,* vol. 31, pp. 1382-1411, 2009.

[34] M. P. Raju and S. Khaitan, "Domain decomposition based high performance parallel computing," *arXiv preprint arXiv:0911.0910,* 2009.

[35] H. A. Van der Vorst, "Bi-CGSTAB: A fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems," *SIAM Journal on scientific and Statistical Computing,* vol. 13, pp. 631-644, 1992.

[36] L. T. Yang and R. P. Brent, "The improved BiCGStab method for large and sparse unsymmetric linear systems on parallel distributed memory architectures," in *Algorithms and Architectures for Parallel Processing, 2002. Proceedings. Fifth International Conference on,* 2002, pp. 324-328.

[37] Y. Saad and H. A. Van Der Vorst, "Iterative solution of linear systems in the 20th century," *Journal of Computational and Applied Mathematics,* vol. 123, pp. 1-33, 2000.

VITA

Abdullah Al Farooq

Department of Modeling, Simulation and Visualization Engineering

1300 E. V. Williams Engineering and Computational Sciences Building

Norfolk, VA 23529

Abdullah Al Farooq has an undergraduate-level background in Computer Science and Engineering from Bangladesh University of Engineering and Technology, Dhaka, Bangladesh, with an emphasis on artificial intelligence. He has completed a Masters degree in Modeling and Simulation at the Department of Modeling, Simulation and Visualization Engineering (MSVE) at Old Dominion University, Norfolk, VA. Abdullah has worked as a graduate assistant for the MSVE department from Fall 2012 to Fall 2013.

DEGREES

Bachelor of Science, Computer Science and Engineering, Bangladesh University of Engineering and Technology, Dhaka, Bangladesh.

Master of Science, Modeling and Simulation, Old Dominion University, Norfolk, VA 23529.