APPROXIMATING SHORTEST PATHS IN LARGE-SCALE NETWORKS WITH AN APPLICATION TO INTELLIGENT TRANSPORTATION

Yu-Li Chou
Robert L. Smith
Department of Industrial & Operations Engineering
The University of Michigan
Ann Arbor, Michigan 48109-2117

and

H. Edwin Romeijn
Rotterdam School of Management
Erasmus University Rotterdam
The Netherlands

Technical Report 95-21

November 1995

Approximating Shortest Paths in Large-scale Networks with an Application to Intelligent Transportation Systems*

Yu-Li Chou† H. Edwin Romeijn‡ Robert L. Smith§

November 22, 1995

Abstract

We propose a hierarchical algorithm for approximating all pairs of shortest paths in a large scale network. The algorithm begins by extracting a high level subnetwork of relatively long links (and their associated nodes) where routing decisions are most crucial. This high level network partitions the shorter links and their nodes into a set of lower level subnetworks. By fixing gateway nodes within the high level network for entering and exiting these subnetworks, a computational savings is achieved at the expense of optimality. We explore the magnitude of these tradeoffs between computational savings and associated errors both analytically and empirically with a case study of the Southeastern Michigan traffic network. An order of magnitude drop in computational times was achieved with an on-line route guidance simulation at the expense of a 5% increase in expected trip times.

1 Introduction

Our interest in this paper directed toward solving very large scale shortest path problems, motivated by the problem of finding minimum travel time paths within an on-line route

*This work was supported in part by the Intelligent Transportation Systems Research Center of Excellence at the University of Michigan.

†Department of Industrial and Operations Engineering, The University of Michigan, Ann Arbor, Michigan 48109-2117; e-mail: yuli.chou@umich.edu.

‡Rotterdam School of Management, Erasmus University Rotterdam, P.O. Box 1738, NL-3000 DR Rotterdam, The Netherlands; e-mail: EROMEIJN@fac.fbk.eur.nl.

§Department of Industrial and Operations Engineering, The University of Michigan, Ann Arbor, Michigan 48109-2117; e-mail: rlsmith@umich.edu.

guidance system. Route guidance within the context of Intelligent Transportation Systems, or ITS, is the task of providing routes between origins and destinations that promise to minimize the trip times experienced. The link travel times that are provided as an input to this function are times dependent forecasts based upon current and anticipated traffic congestion (Kaufman and Smith [7], Kaufman, Wunderlich and Smith [11]). Because of rapid change of link travel times caused by time-varying travel demands and lane blockage resulting from incidents, the data used in computing the shortest paths information is updated periodically, ideally every 5 to 10 minutes. During the time interval between data updates, a shortest path must be provided for every O/D (origin/destination) associated with trips that begin during that time slice. Thus, the calculation of shortest paths must be efficient enough in order to respond in a timely way to trip requests on a real-time basis. Since a realistic problem may have hundreds of thousands of nodes, all of which may be potential origins and destinations, a fast heuristic which can provide good approximations in a limited amount of time may be preferred to exact methods.

We present here a hierarchical approach for finding approximate solutions for shortest path problems. A key idea behind our development is the imposition of a hierarchical network structure. Our approach is motivated by the observation that traffic networks have a hierarchical structure that divides links (and nodes corresponding to intersections of these links) into two or more classes (Yagyu, Ueyama and Azuma [12]). The higher level corresponds to longer links (i.e. highways) where the frequency of decision opportunities is low with routing decisions being correspondingly more important. The lower level corresponds to links of limited duration (i.e. surface streets) and correspondingly more opportunities to correct routing decision errors. The links and nodes comprising the higher level subnetwork corresponds to a macroscopic representation of the whole network. This high level subnetwork partitions the original network into a series of subnetworks at the low level. By solving for all pairs of shortest paths in the higher level subnetwork and interfacing with gateways into the lower level subnetworks, we achieve significant economics of computation, albeit at the expense of a loss of optimality. We explore in this paper the magnitudes of the computational gains and associated errors from optimality.

We begin with introducing the hierarchical algorithm (*HA*) in section 2. We show how the nature of traffic networks induces a natural choice for how the hierarchical network should be modeled. In section 3, we discuss the issue of the efficiency of *HA* for solving various shortest path problems. Furthermore, conditions for efficiently implementing *HA* are presented. In section 4, the efficiency of *HA* when applied to an on-line route guidance system is investigated. A numerical experiment is reported in section 5 which is based upon a real traffic network, the south-eastern Michigan road network.

2 A Hierarchical Approach

2.1 Overview

In this section we introduce a hierarchical approach to solving for shortest paths in a large-scale network. The main idea is to decompose the network into several smaller (low-level) subnetworks. When a shortest path needs to be found between two nodes in the same subnetwork, we approximate this shortest path by the shortest path having the property that *all* nodes on the path are contained in that same subnetwork. The resulting path is an approximative shortest path in that the true shortest path could leave the subnetwork at some point, and re-enter it before arriving at the destination.

To provide approximations of shortest paths between nodes *not* contained in the same subnetwork, we define a (high-level) network (which is actually a subnetwork of the original network) whose nodes intersect all (low-level) subnetworks. The nodes that are present in both the high-level network and in one or more low-level subnetworks are called *macronodes*. The high-level network is called the macronetwork, reflecting the fact that it gives a macroscopic view of the original network. Correspondingly, the low-level subnetworks are called micro-subnetworks, reflecting the fact that it gives a microscopic view of a part of the original network. Given this decomposition, we can approximate a shortest path between two arbitrary nodes of the original network by constraining the path to pass out of the micro-subnetwork containing the origin, through the macronetwork, and into the micro-subnetwork containing the destination. Variants of this general procedure are considered based upon the departing macronode out of the origin subnetwork and the entering macronode into the destination subnetwork chosen.

2.2 Hierarchical Algorithms

Let $G = (V, A, C)$ be a strongly connected directed graph with a set of nodes V, a set of arcs $A \subseteq V \times V$, and a non-negative arc length function $C : A \rightarrow \mathbb{R}$. (A directed graph is strongly connected if there exists a directed path from each node to each node of the network). Our objective is to approximate shortest paths within the network G.

In its most general form, the Hierarchical Algorithm (HA) consists of the following three phases:

Phase 0: Decomposition

Extract a strongly connected *macronetwork* $\tilde{G} = (\tilde{V}, \tilde{A}, \tilde{C})$, where $\tilde{V} \subseteq V$, $\tilde{A} \subseteq \tilde{V} \times \tilde{V}$, and $\tilde{C} : \tilde{A} \rightarrow \mathbb{R}$. The nodes in $\tilde{V}$ will be called *macronodes*, and correspondingly the arcs in $\tilde{A}$ will be called *macroarcs*. Each macroarc $(I, J) \in \tilde{A}$ corresponds to a directed *micropath* from I to J in the original network G where the corresponding arc length $\tilde{C}(I, J)$ is defined as the length of this path. (Note that all macronodes are micronodes as well and that there

exist such a subnetwork associated with every choice of macronode set $\tilde{V} \subseteq$ since G is by assumption strongly connected.)

Divide the original network G into K strongly connected, but not necessarily disjoint, *micro-subnetworks* $G_k = (V_k, A_k, C_k)$, where $V_k \subseteq V$, $A_k = A \cap (V_k \times V_k)$, and $\tilde{V} \cap V_k \neq \emptyset$ for $k = 1, \ldots, K$. Note that computation of every cover $(V_k, k = 1, 2, \ldots, k)$ of V is allowed as long as each V_k in the cover contains at least one macronode.

Phase I: Constrained Shortest Paths

Find all shortest paths in the macronetwork $\tilde{G}$ and in the micro-subnetworks G_k, $k = 1, \ldots, K$. Let the function

$$\tilde{f} : \tilde{V} \times \tilde{V} \rightarrow I\!R$$

return the shortest path lengths in the macronetwork $\tilde{G}$. Similarly, let

$$f_k : V_k \times V_k \rightarrow I\!R$$

return the shortest path lengths in micro-subnetwork G_k, $k = 1, \ldots, K$.

Phase II: Combination

Consider a pair of micronodes $i \in V_k$, $j \in V_\ell$. If $k = \ell$ (i.e., the micronodes are contained in the same micro-subnetwork), then the corresponding shortest path between i and j is approximated by the shortest path from i to j computed in Phase I with all intermediate nodes contained in this subnetwork V_k. Otherwise, the shortest path is approximated by combining three parts computed in Phase I:

(i) the shortest path (in G_k) from i to *some* macronode $I \in V_k \cap \tilde{V}$;

(ii) the shortest path (in the macronetwork G) from I to some macronode $J \in V_\ell \cap \tilde{V}$; and

(iii) the shortest path (in G_ℓ) from macronode J to j.

Note that there always exists such macronodes I and J in steps (i) and (iii) that are also micronodes in G_k and G_ℓ respectively by construction in Phase 0. Variants of *HA* are depend upon how these nodes I and J are selected when there are multiple candidates. Note also that the combination phase can be simplified somewhat when the origin node i and/or the destination node j are macronodes. In particular, in those cases we can skip step (i) and/or (iii) in Phase II. However, depending on the particular choice of macronode made in these steps, it may be fruitful to treat a origin or destination macronode as any other micronode. We will come back to this later when discussing strategies for choosing the exiting and entering macronetwork macronodes for a given origin/destination pair.

2.2.1 Comments on the Decomposition Phase

In the decomposition phase of the Hierarchical Algorithm, there are obviously many possible ways of choosing the macronodes, macroarcs, and micro-subnetworks. However, in a specific application, there usually is a natural hierarchy within the arcs. For example, in a traffic network, the macronetwork can be chosen to be the network consisting of highways and freeways, with the macronodes being a suitably chosen subset of the entrances to and exits from these. The macroarcs will then be micro-paths consisting only of highways and freeways, and the micro-subnetworks could be chosen in a natural way as the subnetworks enclosed by the macroarcs. In section 5 we will illustrate this procedure of forming the macro- and micro-networks using the actual Southeast Michigan road network [9].

2.2.2 Comments on the Combination Within Phase

If, in the combination phase, different micro-subnetworks containing the origin and destination contain only one macronode each, it is unambiguous as to how to construct the approximating path from the description of *HA*. However, often there will be more than one macronode which can be used to connect the micro-subnetworks to the macronetwork. In that case we need to make a choice among these macronodes. An intuitively appealing choice would be to choose the macronodes that allow as to enter the macronetwork as soon as possible. In other words, we choose the closest macronode that can be reached from the origin, and the macronode from which the destination can be reached as quickly as possible. More precisely, if $i \in V_k$ is the origin, and $j \in V_\ell$ $(\ell \neq k)$ is the destination, then the connecting macronodes are chosen to be

$$I^* = \arg \min_{I \in V_k \cap \tilde{V}} f_k(i, I)$$

and

$$J^* = \arg \min_{J \in V_\ell \cap \tilde{V}} f_\ell(J, j).$$

The version of *HA* corresponding to this rule for the selection of macronodes will be called *Nearest HA*.

Obviously the best possible selection rule among all *HA* variants in terms of quality of the solution obtained (but also the most expensive one in terms of computation time) is to select the pair of macronodes that yields the shortest approximate path. In other words, choose the pair of connecting macronodes as follows:

$$(I^*, J^*) = \arg \min_{(I,J) \in (V_k \cap \tilde{V}) \times (V_\ell \cap \tilde{V})} \left\{ f_k(i, I) + \tilde{f}(I, J) + f_\ell(J, j) \right\}.$$

This variant of *HA* will be called *Best HA*.

At times there will be more than one micro-subnetwork containing the origin (and/or destination). In that case we would adjust the choice of macronodes as follows. For *Nearest HA*, we set:

$$I^* = \arg \min_{I \in \cup_{k:i \in V_k} V_k \cap \tilde{V}} f_k(i, I)$$

and similarly for J^*. Similar adjustments would be made for finding (I^*, J^*) for *Best HA*.

3 Computational Complexity Analysis

For the heuristic *HA* to be useful, there obviously needs to be time savings to balance the loss in precision in computing the approximate shortest paths produced by it when compared to an exact algorithm. In this section we investigate the relative efficiency of both *Nearest HA* and *Best HA* under various conditions. Before analyzing the complexity of *HA* we first review the concepts of $\mathcal{O}$, Ω, and Θ functions (see e.g. Aho, Hopcroft and Ullman [1]).

Definition 3.1 *Let f and g be functions from the non-negative integers to the non-negative reals.*

(a) $f(n)$ is said to be $\mathcal{O}(g(n))$ (or $f(n) = \mathcal{O}(g(n))$) if there exist positive constants c and n_0 such that

$$f(n) \leq cg(n) \qquad \text{for all } n \geq n_0.$$

Note that the $\mathcal{O}$-notation is used to specify an upperbound on the rate of growth *of some function. Analogously, the Ω-notation is used to provide a lowerbound on the* rate of growth, *where as the Θ-notation is used if the upper- and lowerbounds coincide.*

(b) The function $f(n)$ is said to be $\Omega(g(n))$ (or $f(n) = \Omega(g(n))$) if there exist positive constants c' and n_0 such that

$$f(n) \geq c'g(n) \qquad \text{for all } n \geq n_0.$$

(c) The function $f(n)$ is said to be $\Theta(g(n))$ (or $f(n) = \Theta(g(n))$) if there exist positive constants c, c' and n_0 such that

$$c'g(n) \leq f(n) \leq cg(n) \qquad \text{for all } n \geq n_0.$$

Now, assume that the number of nodes $|V|$ in the graph G, is $\Theta(N)$, where N is an input parameter characterizing the size of the graph. Since we are focusing on road networks, which are sparse, the most efficient algorithm for computing all shortest path lengths exactly is Dijkstra's algorithm (see e.g. Denardo [5], Dreyfus and Law [6]). The number of operations needed by this algorithm is $\mathcal{O}(N \log N)$ and $\Omega(N)$ for solving for the shortest paths from a

single origin to all destinations. We will repeatedly make use of this result in the remainder of this section.

To gain insight into the potential gain in computation time achievable by *HA* and how we should decompose the original network to realize this gain, we assume within this section that we decompose the network in such a way that the number of macronodes is $\Theta(N^m)$ (for some $0 < m < 1$), and that the number of micro-subnetworks is $\Theta(N^k)$ (for some $0 < k < 1$). (The actual performance gain under real world conditions will be explored in section 5.) Each micro-subnetwork is further assumed to be of roughly the same size, so that the number of nodes in each subnetwork is $\Theta(N^{1-k})$. Moreover, we assume that each macronode is shared, on average, by a constant number of micro-subnetworks, so that the number of macronodes per micro-subnetwork is $\Theta(N^{m-k})$. Since we need at least 1 macronode per micro-subnetwork, we obviously need that $k \leq m$. The relative magnitudes of m and k to achieve the greatest computational savings will be explored under two implementation scenarios in the next two subsections.

3.1 All-Pairs Shortest Path Problem

We begin by investigating the efficiency of *HA* for solving the all-pairs shortest path problem. Let $C_N(N)$ denote the number of operations needed for approximating *all* shortest path lengths in a network structure defined above using *Nearest HA*, $C_B(N)$ the corresponding number using *Best HA*, and $C_D(N)$ the number of operations needed for computing all shortest paths using Dijkstra's algorithm in the same network. The following theorem gives the relative efficiency of *Nearest HA* with respect to Dijkstra's algorithm.

Theorem 3.2 *The relative efficiency of* Nearest HA *with respect to Dijkstra's algorithm for the all-pairs shortest path problem satisfies*

$$\frac{C_D(N)}{C_N(N)} = \mathcal{O}(\log N)$$

and

$$\frac{C_D(N)}{C_N(N)} = \Omega(1).$$

Proof: See Appendix. ∎

From the above result it is clear that the combination phase is the most time-consuming one, and, in fact, that the complexity for that phase is *independent of the particular decomposition chosen.* Notwithstanding this fact, we could try to minimize the effort necessary for Phase I, even though this phase is already much less time-consuming than Phase II. Recall from the proof of theorem 3.2 that Phase I takes on the order of $N^{\max(2-k,2m)}$ operations (disregarding

the 'log N' term). Recall also that $m \geq k$. Therefore, it seems optimal (with respect to computation time) to choose $m = k$. The computation time for Phase I then becomes of the order $N^{\max(2-k,2k)}$, which is minimized if $2 - k = 2k$, or if $k = \frac{2}{3}$. This means that to most efficiently implement *Nearest HA*, the network should be decomposed in such a way that

(i) the number of subnetworks should be of the same order as the number of macronodes, i.e. the number of macronodes per subnetwork should be independent of the size of the network

(ii) the size of each of the subnetworks should be smaller than the size of the macronetwork.

However, note that a pure minimization of the computational effort does not take into account any natural structure present in the network, or the magnitude of the error incurred by the corresponding decomposition.

The following theorem is the analogue of theorem 3.2 for *Best HA*.

Theorem 3.3 *The relative efficiency of* Best HA *with respect to Dijkstra's algorithm for the all-pairs shortest path problem satisfies*

$$\frac{C_D(N)}{C_B(N)} = \mathcal{O}(N^{2(k-m)} \log N)$$

and

$$\frac{C_D(N)}{C_B(N)} = \Omega(N^{2(k-m)}),$$

where $m \geq k$.

Proof: See Appendix. ■

Unlike in the case of *Nearest HA*, the *only* instances where savings might be obtained occur when $k = m$, i.e. *when the number of macronodes per subnetwork does not grow with* N. Whenever $m > k$, the complexity 'savings' actually become a complexity 'dissavings,' in the sense that Dijkstra's algorithm is *more efficient* than *Best HA*! However, if $k = m$, then the relative efficiency of *Best HA* is the same as the relative efficiency of *Nearest HA*.

3.2 Limited Shortest Path Problem

We have seen that *HA* does not always perform more efficiently than Dijkstra's algorithm for solving all shortest paths in a network defined as above. Fortunately, however, in traffic networks we rarely need to compute the shortest paths between *all* pairs of nodes since network data is being updated periodically and not all possible trips will take place between any two updates (see section 4). To model this observation, we assume that there are $\Theta(N^\gamma)$

nodes which can occur as origins or destinations. Moreover, we assume that these origins (destinations) are spread out uniformly over the network, and that every micro-subnetwork contains at least one origin, so that $\gamma \geq k$.

Similar to the notation in the previous section, let $C_D(N;\gamma)$, $C_N(N;\gamma)$, and $C_B(N;\gamma)$ denote the number of operations needed for computing or approximating the required shortest paths in the network structure defined above using Dijkstra's algorithm, *Nearest HA*, and *Best HA*, respectively. In particular, it is easy to see that

$$C_D(N;\gamma) = \mathcal{O}(N^{1+\gamma}\log N)$$

and

$$C_D(N;\gamma) = \Omega(N^{1+\gamma}).$$

Theorem 3.4 *The relative efficiency of* Nearest HA *with respect to Dijkstra's algorithm for the limited-pairs shortest path problem satisfies*

$$\frac{C_D(N;\gamma)}{C_N(N;\gamma)} = \mathcal{O}(N^{\ell_1(k,m,\gamma)}\log N)$$

and

$$\frac{C_D(N;\gamma)}{C_N(N;\gamma)} = \Omega\left(\min(N^{\ell_2(k,m,\gamma)}/\log N, N^{1-\max(\gamma,m-k)})\right),$$

where $\ell_1(k,m,\gamma) = \min(k-\max(0,m-\gamma), 1-\gamma-2\max(0,m-\gamma))$ *and* $\ell_2(k,m,\gamma) = \min(k-\max(0,m-\gamma), 1-\gamma-2(m-\gamma))$.

Proof: See Appendix. ∎

As before, we can investigate the optimal number of macronodes and micro-subnetworks in terms of computational efficiency. If, for simplicity, we only consider the $\mathcal{O}$-expression for the relative efficiency of *Nearest HA*, we see that

$$\frac{C_D(N;\gamma)}{C_N(N;\gamma)} = \mathcal{O}(N^{\min(k,k-m+\gamma,1-\gamma,1+\gamma-2m)}\log N).$$

Clearly, it is optimal to choose k as large as possible. Since $k \leq m$ by assumption, we will let $k = m$. The relative efficiency then becomes

$$\frac{C_D(N;\gamma)}{C_N(N;\gamma)} = \mathcal{O}(N^{\min(m,\gamma,1-\gamma,1+\gamma-2m)}\log N).$$

The optimal relative efficiency can now be obtained if $m = 1+\gamma-2m$, or $m = (1+\gamma)/3$. It is easy to see that $(1+\gamma)/3 \geq \min(\gamma, 1-\gamma)$, so that the optimal relative efficiency is $\mathcal{O}(N^{\min(\gamma,1-\gamma)}\log N)$, for $k = m = (1+\gamma)/3$.

Now, consider the case of *Best HA*, the following theorem is the analogue of theorem 3.4.

Theorem 3.5 *The relative efficiency of* Best HA *with respect to Dijkstra's algorithm for the limited-pairs shortest path problem satisfies*

$$\frac{C_D(N;\gamma)}{C_B(N;\gamma)} = \mathcal{O}(N^{\min(k-\max(0,m-\gamma),1-\gamma-2(m-k))} \log N)$$

and

$$\frac{C_D(N;\gamma)}{C_B(N;\gamma)} = \Omega\left(\min(N^{\min(k-\max(0,m-\gamma),1+\gamma-2m)}/\log N, N^{1-\gamma-2(m-k)})\right).$$

Proof: See Appendix. ■

From Theorem 3.5, we have the following relative efficiency for the case of *Best HA*

$$\frac{C_D(N;\gamma)}{C_B(N;\gamma)} = \mathcal{O}(N^{\min(k,k-m+\gamma,1-\gamma-2m+2k)} \log N).$$

Again, it is optimal to choose $k = m$, yielding

$$\frac{C_D(N;\gamma)}{C_B(N;\gamma)} = \mathcal{O}(N^{\min(m,\gamma,1-\gamma)} \log N).$$

So the optimal relative efficiency is again $\mathcal{O}(N^{\min(\gamma,1-\gamma)} \log N)$, attained for $k = m \geq \min(\gamma, 1-\gamma)$.

4 On-Line Shortest Path Calculations

In this section, we explore the complexity of *Best HA* and Dijkstra's algorithm when implemented in an on-line shortest path route guidance system. In an on-line route guidance system, the arc length function C is updated regularly, on the basis of recent information concerning the link traversal times (see Kaufman and Smith [7]). In such a system, a path will have to be provided for a certain O/D-pair if a request is made for that pair during the *time slice* during which the arc length function is valid. Moreover, such a path will only have to be computed if and when first time the O/D-pair is requested during a time slice. Unlike the problems discussed in the preceding section, where the number of O/D pairs requiring solutions was assumed known, the relevant question here is: "How many different O/D-pairs will be requested in a given time slice?". To help answer this question, we first describe a classical problem, called the *coupon-collector problem.*

4.1 Coupon-Collector Problem

The coupon-collector problem (CCP) is defined as follows. Suppose a set containing ℓ distinct objects is given. A collector samples from the set with replacement, where with each trial he has a probability p_i of drawing the i-th object, independently of previous events.

Given a probability vector p, let $Y_n(p)$ be the number of different items observed in the first n trials. The following results are well-known (see e.g. Boneh and Hofri [3], Caron and McDonald [4]):

$$E(Y_n(p)) = \sum_{i=1}^{\ell} (1 - (1 - p_i)^n) . \tag{1}$$

4.2 Computational Complexity Per Time Slice

In this subsection, we investigate the complexity of *Best HA*, as well as the Dijkstra algorithm, in an on-line route guidance system. For simplicity, we assume that each micro-subnetwork has the same size, i.e. $|V_k| = |V|/K$ for all k. We start by introducing some additional notation.

Let the expected number of O/D-requests per unit time be constant, given by n. If the length of the time slice is equal to τ, then the expected number of O/D-requests in a time slice is $n\tau$. We also assume that the probabilities for the O/D-requests are homogeneous over time. In particular, let p_{ij} denote the probability that an O/D-request corresponding to the pair of nodes (i, j) is made, for all $i, j \in V$. Furthermore, let $p_{i\cdot}$ be the probability that an O/D-request is made from origin i, i.e. $p_{i\cdot} = \sum_j p_{ij}$. Similarly, let the probability that an O/D-request is made to destination j be denoted by $p_{\cdot j} = \sum_i p_{ij}$. If $W \subset V$ is a set of nodes, denote the probability that an O/D-pair is requested from an origin in W by $p_{W\cdot} = \sum_{i \in W} p_{i\cdot}$, and denote the probability that a request is made to a destination in W by $p_{\cdot W} = \sum_{j \in W} p_{\cdot j}$. Finally, let $\mathcal{V}_i = \cup_{k: i \in V_k} V_k$ be the union of nodes contained in the micro-subnetworks which share node $i \in V$.

4.2.1 Complexity of Dijkstra's Algorithm

We now consider the expected number of computations required by the Dijkstra algorithm per time slice. We may recall that the shortest paths from an origin, say i, to all the nodes in the network G can be obtained at modest additional cost (and the same complexity!) by using Dijkstra's algorithm, even if only one O/D-path is desired. Therefore, we assume that we compute all shortest paths from a single origin the first time an O/D-request is made from that origin during the time slice.

This problem can clearly be viewed as an instance of the coupon-collector problem. The items are all possible origins $i \in V$ that can be requested, with associated probabilities $p_{i\cdot}$, and each O/D-request within a time slice is an independent trial. Therefore, from equation

(1), the expected number of one-pair shortest path problems we have to solve over the network G is

$$\sum_{i=1}^{|V|}(1-(1-p_{i\cdot})^{n\tau})$$

per time slice. Thus, the computational complexity of Dijkstra's algorithm for the duration of τ time units can be obtained by multiplying the expected number of different origins observed during a time slice with the complexity of the one-pair shortest path problem, yielding

$$\mathcal{T}_D(\tau)=\mathcal{O}\left(|V|\log|V|\cdot\sum_{i=1}^{|V|}(1-(1-p_{i\cdot})^{n\tau})\right). \tag{2}$$

4.2.2 Complexity of *HA*

To investigate the complexity of the two versions of *HA*, we distinguish, as we did before, between the number of operations required for Phase I and Phase II computations.

In Phase I, several shortest path problems over different subnetworks need to be solved. These problems can be classified as: 1. the shortest paths in the macro-network, 2. the shortest paths within a subnetwork corresponding to origins, 3. the shortest paths within a subnetwork corresponding to destinations. We first consider the expected number of computations required for solving the shortest paths in the macro-network per time slice. During a time slice, a one-to-all shortest paths problem will be solved in the macro-network from each macronode that is an element of the subnetwork containing the origin of some O/D-request that was made during that time slice. In terms of the coupon-collector problem, the items are now the macronodes, and the item probability corresponding to macronode $I\in\tilde{V}$ is

$$q_I=\sum_{i\in\mathcal{V}_I}\sum_{j\notin\mathcal{V}_i}p_{ij}.$$

The expected number of corresponding one-origin shortest path problems that has to be solved during time slice τ is

$$\sum_{I\in\tilde{V}}(1-(1-q_I)^{n\tau}).$$

The product of the expected number of one-pair shortest path problems and the cost for solving each problem brings the complexity of solving the shortest paths in the macro-network to

$$\mathcal{O}\left(|\tilde{V}|\log|\tilde{V}|\cdot\sum_{I\in\tilde{V}}(1-(1-q_I)^{n\tau})\right). \tag{3}$$

We next consider the complexity of solving for the second type problem, i.e. all shortest paths from an origin, within the subnetwork containing that origin, corresponding to requests

during a time slice. In other words, for each new origin, the shortest paths from that origin to all destinations within its subnetwork are computed. In this case, the items are the origins, with corresponding probability $p_{i\cdot}$ as defined above. The complexity for this phase over the time slice becomes

$$\mathcal{O}\left((|V|/K)\log(|V|/K)\cdot\sum_{i=1}^{|V|}(1-(1-p_{i\cdot})^{n\tau})\right). \tag{4}$$

For the type three problem, it can be seen that the one-pair shortest path problem from a macronode within a given subnetwork containing that macronode is computed the first time a destination of an O/D-request is in that subnetwork, and the origin is not. For each O/D-request, the probability that the destination is an element of subnetwork G_k (the item probability) is

$$\rho_k = \sum_{i\notin V_k}\sum_{j\in V_k} p_{ij}.$$

If a destination in G_k is sampled, the shortest paths to that destination from all macronodes in G_k needs to be computed. Since there are on the order of $|\tilde{V}|/K$ macronodes in each subnetwork (recall that the number of subnetworks sharing a given macronode is roughly a constant), the total computational cost for these computations during the time slice is

$$\mathcal{O}\left((|V|/K)\log(|V|/K)\cdot(|\tilde{V}|/K)\cdot\sum_{k=1}^{K}(1-(1-\rho_k))^{n\tau})\right). \tag{5}$$

The complexity of Phase I for τ units of time, denoted by $\mathcal{T}^{\mathrm{I}}(\tau)$, can now be obtained by adding equations (3)–(5).

In Phase II, the combination is performed for each new appearance of an O/D-pair across subnetworks. For *Best HA*, on the order of $(|\tilde{V}|/K)^2$ feasible paths have to be compared for each O/D-pair. If we define

$$\delta_{ij} = \begin{cases} 1 & \text{if } i \text{ and } j \text{ are in different subnetworks} \\ 0 & \text{otherwise.} \end{cases}$$

for all $i, j \in V$, then the complexity of Phase II computations for *Best HA* during the time slice is

$$\mathcal{T}_B^{\mathrm{II}}(\tau) = \mathcal{O}\left((|\tilde{V}|/K)^2\cdot\sum_{i=1}^{|V|}\sum_{j=1}^{|V|}(1-(1-p_{ij}\cdot\delta_{ij})^{n\tau})\right). \tag{6}$$

The total complexity of *Best HA* for the duration of a time slice τ, denoted by $\mathcal{T}_B(\tau)$, is then simply the equal to

$$\mathcal{T}_B(\tau) = \mathcal{T}^{\mathrm{I}}(\tau) + \mathcal{T}_B^{\mathrm{II}}(\tau). \tag{7}$$

It is clear that the complexity expressions in equations (2) and (7) are extremely difficult to compare. An overall comparison of the efficiency of *HA* with Dijkstra's algorithm cannot be made. Therefore, in the next section, we will return to this topic when we consider an experiment using a set of real-life data.

5 Application to the Southeast Michigan Vehicles Transportation Network

In this section, we illustrate the process of implementing the Hierarchical Algorithms for solving shortest path problems in a large-scale real network. In particular, special attention is paid to the case of on-line route guidance. The methodology of implementing *HA* as well as the numerical results are the main focus of this section.

5.1 Description of the Network and Data Resources

The real network we used to conduct the experiments in this section is the Southeast Michigan road network. This road network was generated from the topological data provided by the regional planning authority of the Southeast Michigan Council of Governments (SEMCOG). The region covered by this road network consisted of the following five counties: Macomb, Monroe, Oakland, Washtenaw, and Wayne County. Moreover, this network included most roadway segments in the region, from freeways to major streets.

There are 3,189 nodes and 5,658 links (11,316 directed arcs) in this road network, where links represent the street segments and nodes represent the intersections. Figure 1 shows the Southeast Michigan road network. Links are classified by SEMCOG into six different levels from class 1 (major freeways) to class 6 (local streets). Each road class has an associated speed limit, as shown in table 1. The link travel time is then defined as the time it takes to traverse the link at the corresponding speed limit.

Road class	1	2	3	4	5	6
Speed limit (miles/hr)	65	50	40	40	35	35

Table 1: *Speed limits.*

Other resources provided by SEMCOG are a set of *zone centroids*, and the average number of trips between zone centroids over a typical 24-hour period. Each zone centroid represents the center of the population within a certain area (zone). Standard transportation methods were used by SEMCOG [9] to identify 1,543 zones and corresponding zone centroids across the road network as origins and destinations for regional travel.

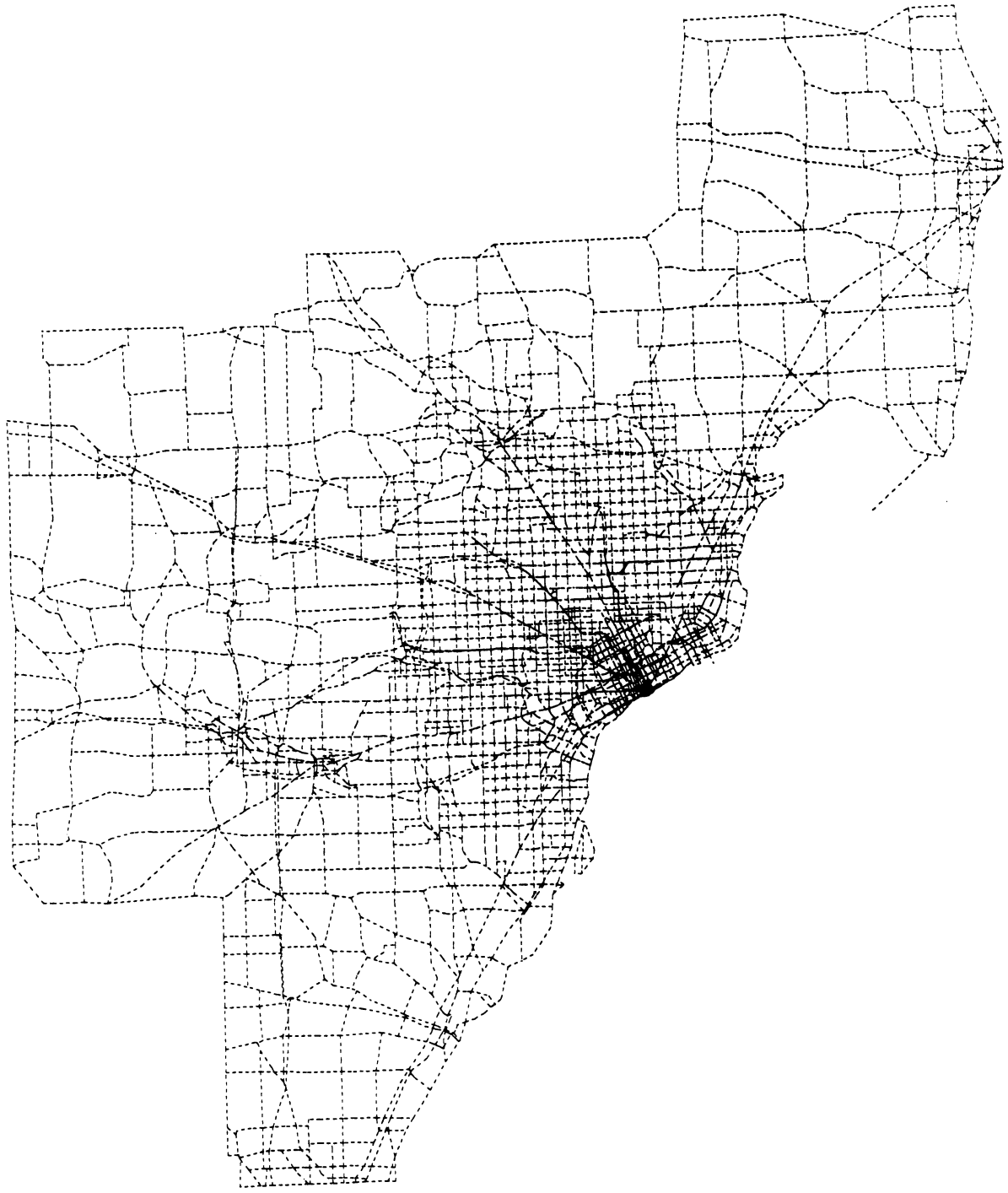

Figure 1: *The Southeast Michigan road network.*

5.2 Aggregation of the Network

In this section, we consider the problem of defining an appropriate two-level structure for the Southeast Michigan network so that it is suitable for the application of *HA*.

We will first form a macro-network in the Southeast Michigan network. As we have suggested in section 2.2.1, a natural way of forming the macro-network for a traffic network is to define the high speed roads and the major interchanges as the macrolinks and the macronodes, respectively. Based upon this principle, we start by defining the nodes connecting either two class 1 links, or one class 1 and one class 2 link as macronodes. The initial macronetwork is then formed by connecting these macronodes through macrolinks formed by a path consisting of class 1 links only. However, the macro-network obtained in this way is not connected, and contains "gaps", which obviously will cause poor performance of the *HA* algorithms. Figure 2 shows the original two-level network where the thick solid lines represent the macrolinks. To solve these problems, we upgrade some lower level links to class 1 (but retaining the original link traversal times, so that this upgrading does not have an effect on the complete network), and form new macrolinks using these new class one links.

The resulting macro-network is shown in Figure 3. Once the macro-network is defined, we decompose the network into a set of micro-subnetworks by letting the macrolinks carve up the network. In this way, each area surrounded by macrolinks is a micro-subnetwork. Figure 4 shows the micro-subnetworks defined for the Southeast Michigan network.

Finally, we add the zone centroids to the network. These zone centroids are isolated nodes in the network. To connect these nodes to the network we add, for each zone centroid, two links which connect the centroid to the two nearest nodes in the network (in Euclidean distance). These centroid connectors are then assigned a length equal to this Euclidean distance, and are characterized as class 7 links with a low speed limit of 20 miles an hour. This models the fact that the travel on the connectors (i.e. the low volume surface streets) between centroids and the arterial network is generally slow. The addition of the zone centroids and their connector links increases the size of the network from 3,189 nodes to 4,732 nodes (and from 11,316 to 14,300 arcs).

5.3 Numerical Results

In this section, we report on computational experiments with both *Basic* and *Best HA*, as well as Dijkstra's algorithm, for solving for shortest paths for all possible *trip pairs* in the Southeast Michigan road network. The heuristics and the Dijkstra algorithm were coded in C and executed on an IBM RS6000 workstation.

As noted before, the only trips possible in the network are between zone centroids. In other words, we have to compute the shortest paths between $1{,}543 \times 1{,}543 = 2{,}380{,}849$ pairs of nodes, in a network with 4,732 nodes. The total number of trips is equal to $n = 12{,}753{,}236$.

Furthermore, the data provided by SEMCOG on the average number of trips between

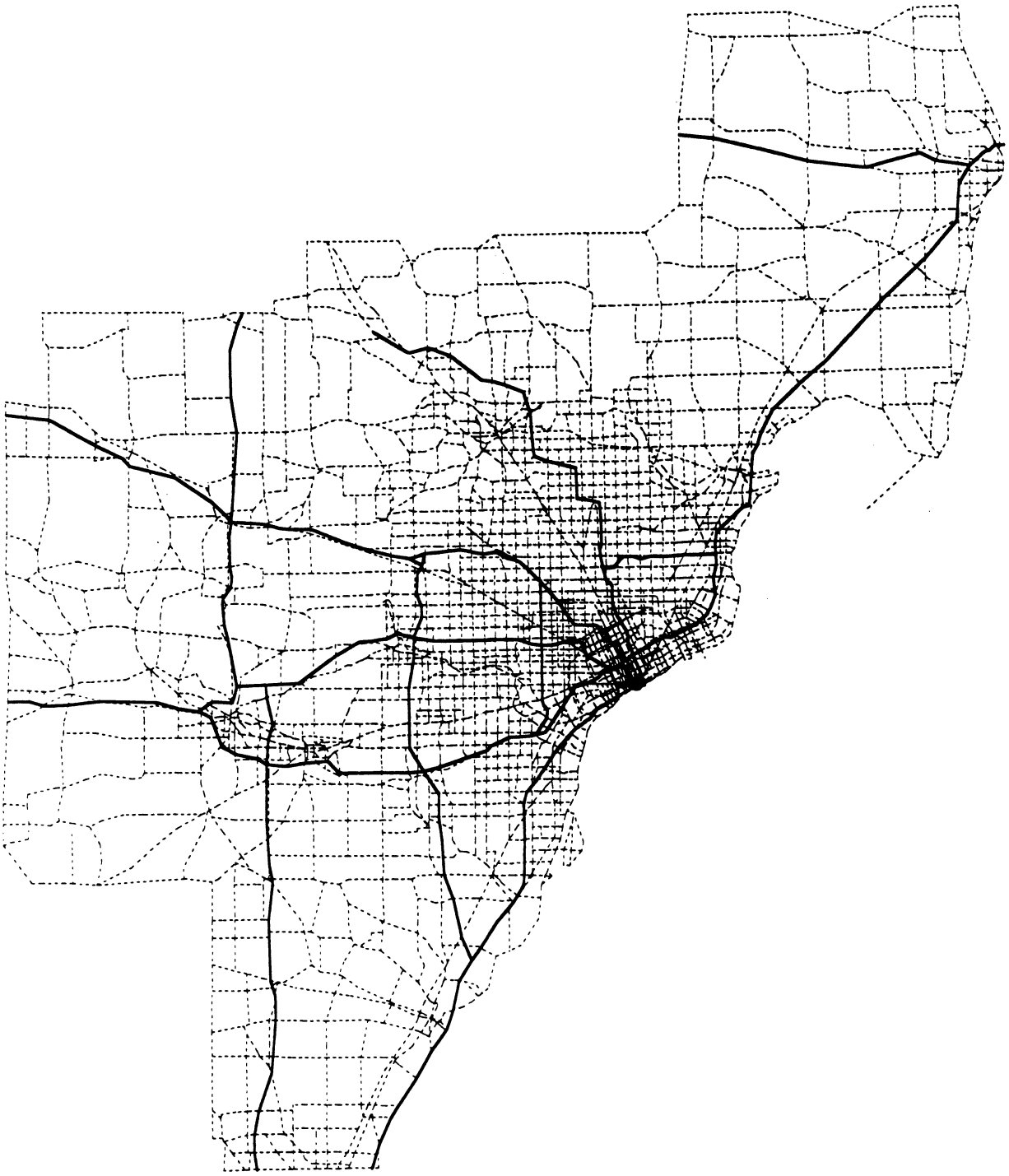

Figure 2: *The original two-level Southeast Michigan road network.*

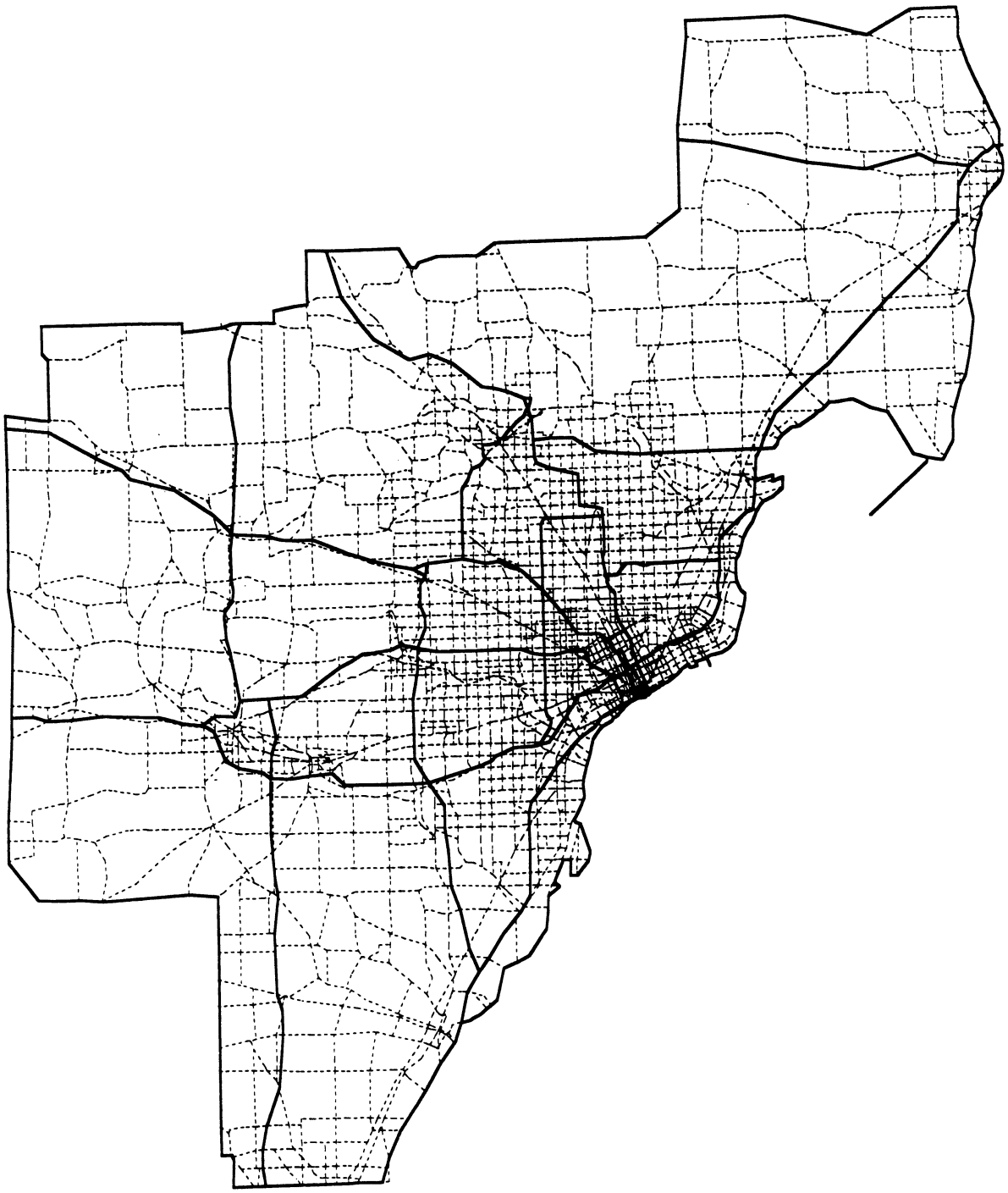

Figure 3: *The modified two-level Southeast Michigan road network.*

Figure 4: *The micro-networks of the Southeast Michigan road network.*

centroids over a 24-hour period is incorporated to weigh the errors resulting from the approximation algorithms. In particular, the error associated with each O/D pair is weighed by the corresponding number of trips made during 24 hours. The average error resulting from *HA* is then calculated as follows:

$$\bar{\epsilon} = \frac{\sum_{i=1}^{n} (\hat{\ell}_i - \ell_i) \cdot n_i}{\sum_{i=1}^{n} \ell_i n_i}$$

where, for O/D-pair i, ℓ_i denotes the shortest path length, $\hat{\ell}_i$ denotes the approximated shortest path length, and n_i denotes the number of trips during a 24-hour period.
The weighted results of solving the shortest paths for all trip pairs are shown in Table 2. We can see from this table that *Nearest HA* is about 15 times as efficient as Dijkstra's algorithm, whereas *Best HA* is almost twice as efficient as Dijkstra's algorithm. Although the error for *Nearest HA* is quite high, the error for *Best HA* is small.

	$\bar{\epsilon}$ (%)	CPU time (seconds)		
		Phase I	Phase II	Total
Dijkstra	0.0	–	–	60.22
Nearest HA	21.5	4.25	0.66	4.91
Best HA	4.9	4.68	29.32	34.00

Table 2: *Results of solving the shortest paths for all trips.*

In the next section, we conduct a simulation of on-line shortest path information queries in the Southeast Michigan road network. Since *Best HA* gives very accurate approximations at acceptable cost, we will only focus on *Best HA* and Dijkstra's algorithm in the next section.

5.4 An Application of On-Line Shortest Path Queries

In this subsection, we conduct a numerical experiment for an on-line route guidance system in the Southeast Michigan road network. In this experiment, O/D-requests are made probabilistically between zone centroids, where the probabilistic O/D-request matrix is generated based on the data of the average number of trips made for a 24-hour period. We assume that the number of O/D-requests is constant over time, and thus the number of requests per minute is equal to 12,753,236/1,440 $\approx$ 8,856. The probability of a request made for O/D pair i, denoted by p_i, is then computed as $p_i = n_i/n$ for all i.

The main purpose of this experiment is to compare the relative efficiency of *Best HA* and Dijkstra's algorithm in an on-line route guidance system through the simulation conducted using real-life data.

5.4.1 CPU Times Required by the Algorithms Per Time Slice

As we have discussed above, in an on-line route guidance system the complexity of Dijkstra's algorithm as well as that of *Best HA* depends on the duration of the time slice that is being considered. Our goal in this simulation is to compare the efficiency of *Best HA* with that of Dijkstra's algorithm using various time slice durations.

In an on-line route guidance system, it is sufficient to only provide the shortest path information for the O/D pair currently being requested. Moreover, recall that, using Dijkstra's algorithm, the shortest path from an origin to a destination node is obtained once the destination node is labeled as a permanent node during the process of building a shortest path tree from that origin. However, at modest additional cost we can obtain the complete shortest path tree from the origin. We propose two policies for implementing Dijkstra's algorithm in an on-line route guidance system based on these properties of Dijkstra's algorithm. The first policy (Policy 1) we propose is to stop the search process as soon as the destination becomes "permanently labeled" when solving for the shortest path for an O/D pair. Thus, under this policy, only the shortest paths from the origin to nodes that have been labeled permanently are computed. The second policy (Policy 2), on the other hand, is to complete the search until all the nodes have been labeled permanent. As a result, the shortest paths from the origin to all other nodes in the network are obtained after the search. When solving for the shortest path for one particular O/D pair, Dijkstra's algorithm obviously performs more efficiently under the first policy since the search process is stopped after obtaining the necessary information. However, when applied to an on-line route guidance system, there is no guarantee of better performance of the first policy. Note that in an on-line route guidance system, the shortest path information, once obtained, is stored until an update of the distance (link travel time) matrix takes place. Thus, it may take longer to complete a single shortest path search from an origin under the second policy, but more shortest path information can be provided at modest additional cost within the current time slice when requests are made from the same origin.

In practice, a reasonable time period between the updates of link traversal times will be no more than 20 minutes. Thus, we run each single simulation for a 20-minute period. Furthermore, the cumulative CPU times required to solve the shortest paths in response to O/D-requests are recorded at different points in time during each simulation. Our major aim is to simulate the CPU times consumed by Dijkstra's algorithm and by *Best HA* per time slice in an on-line route guidance system using various time slice lengths. Figure 5 shows the total CPU times required by both approaches under different policies over the whole simulation period. It can be seen that under either policy, the CPU time required by

using Dijkstra's algorithm increases rapidly within a very short period of time, while that required by *Best HA* increases gradually and slowly throughout the simulation period. The figure also shows that Dijkstra's algorithm performs more efficiently under the first policy if the duration of a time slice is less than 3.7 minutes. However, for a longer period of time, the second policy outperforms the first policy. A similar observation can be made for *Best HA*. However, since only subnetworks are considered instead of the complete network, the critical time slice length where the second policy starts dominating the first one is much smaller (about 1 minute).

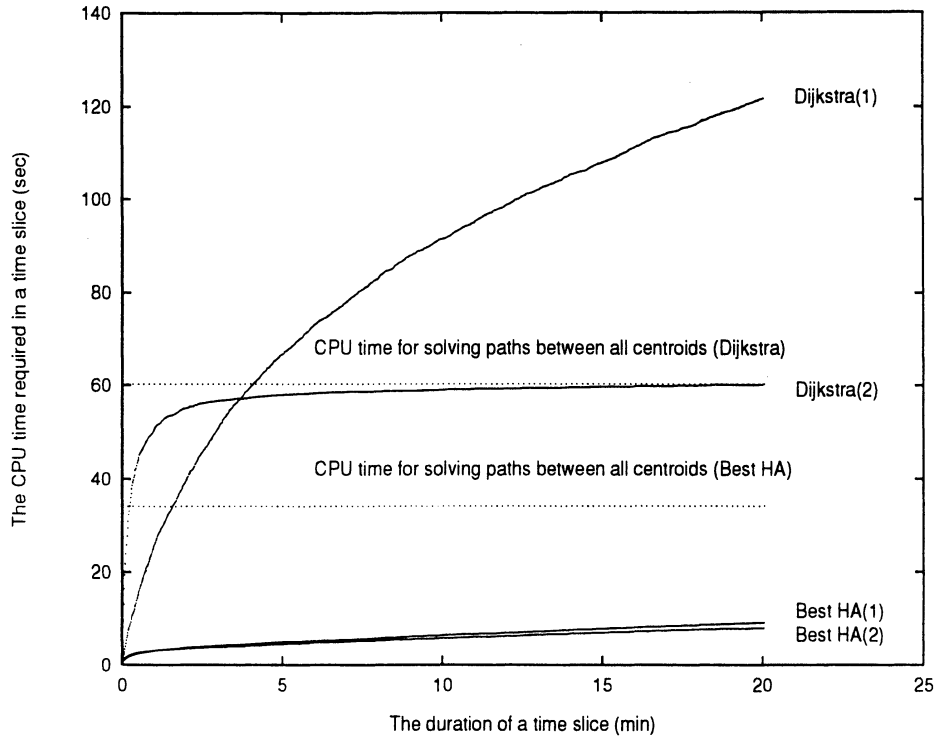

Figure 5: CPU time as a function of time slice length.

We can see from figure 5 that *Best HA* is much more efficient than Dijkstra's algorithm during the whole simulation period. The amounts of time savings achieved by *Best HA* over the Dijkstra algorithm for various periods of time are shown in Figure 6. To be more specific, Table 3 shows the efficiency ratio's for a number of time slices. The average relative error by *Best HA* collected at various points in time during the simulation are shown in Figure 7. For a 20-minute simulation period, the *average relative error is 5.78%*.

5.4.2 Expected CPU Time Required by the Algorithms

In this section, we compute the expected CPU times required by both algorithms, based upon the discussion conducted in section 4.2, for an on-line route guidance system implemented

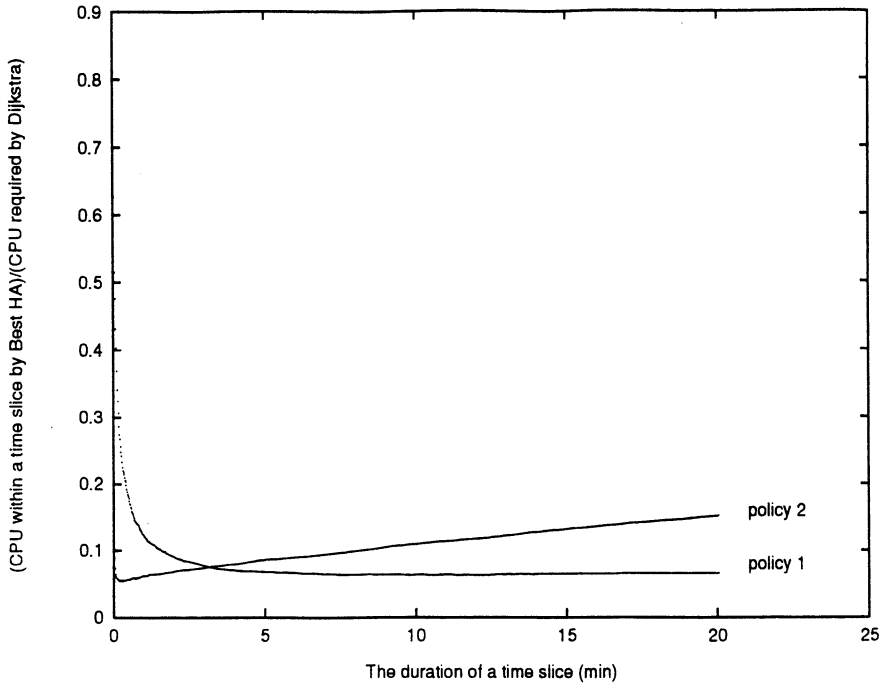

Figure 6: *The CPU time savings achieved by* Best HA *over Dijkstra's algorithm.*

time slice	first policy	second policy
5	14.7	11.6
10	15.8	9.2
15	15.5	7.6
20	15.2	6.6

Table 3: *Efficiency ratio of Dijkstra's algorithm vs.* Best HA.

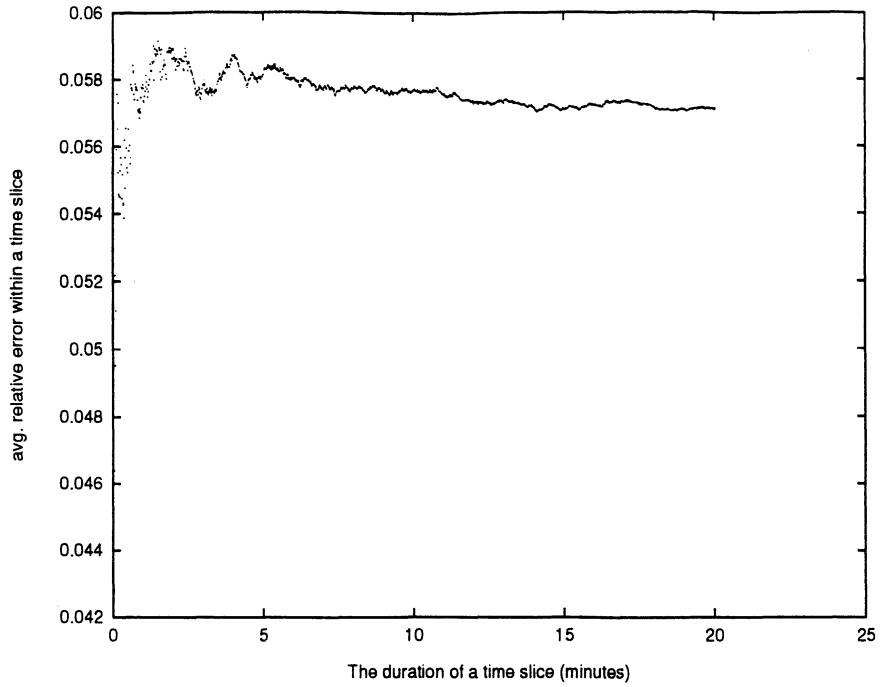

Figure 7: *The average relative error yielded by* Best HA *during the simulation period.*

in the Southeast Michigan road network. We further compare the expected values to the simulation results obtained in the preceding section.

As we have shown in section 4.2, equation (2) gives the expected amount of computational effort required by Dijkstra's algorithm to solve the shortest paths in response to O/D-requests for τ units of time. This expected number is obtained by multiplying the expected computation time required to solve a one-pair shortest path problem over the whole network with the expected number of shortest paths need to be solved in a τ-unit period. We may also recall that by applying *Best HA*, there are four types of subproblems that need to be solved. Furthermore, different computational effort is required to solve different types of subproblems. These subproblems can be categorized as follows:

1. the shortest paths in the macro-network,
2. the shortest paths within a subnetwork corresponding to origins,
3. the shortest paths within a subnetwork corresponding to destinations,
4. the combination of approximate solutions.

Again, the expected CPU time required for solving each type of subproblem can be obtained by multiplying the expected number of subproblems that need to be solved during a time slice with the corresponding time needed for solving a single subproblem. The computation

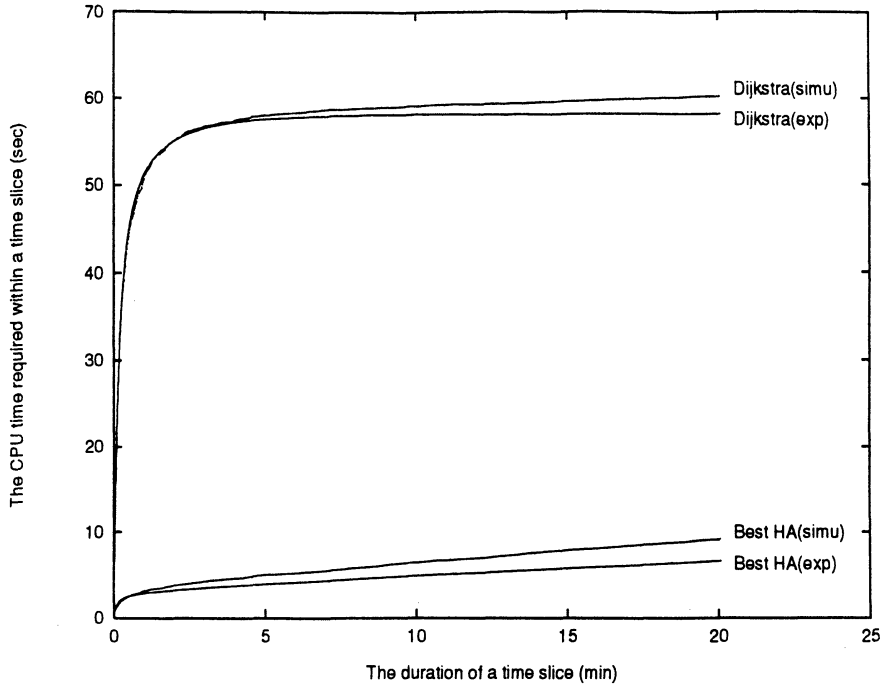

Figure 8: *Expected vs. simulation CPU times.*

time required by *Best HA* is then the sum of the expected times required for solving these subproblems. For more details on the expected computation time required by the algorithms, we refer to equations (3)–(6) in section 4.2.

We compute the expected CPU times required by both algorithms based upon equations (2) and (7). However, we substitute the average CPU time required to solve each single subproblem for the corresponding expected computational effort. The expected CPU times required by both algorithms are then computed using various periods of time. Figure 8 shows the expected total CPU time required by both algorithms compared to the simulation results obtained in the preceding subsection. Furthermore, the expected CPU times required by *Best HA* for solving subproblems of types 1–4 as mentioned above are shown in figure 9, where $CTi(t)$ is the (expected) CPU time corresponding to a subproblem of type i in a t-unit time slice.

Computing the expected CPU times required by the algorithms as illustrated here is a very cost effective way of obtaining an overall picture on the performance of the algorithms in an on-line route guidance system. This is, in particular, very helpful in making decisions on the choices of the algorithm, the duration of a time slice, etc. Although the expected CPU times computed this way might not be as accurate as the simulation results, it gives a rather good estimation without the effort and the computer time and memory space required in conducting a simulation.

The expected computational costs can also be used to adjust the size of the macro-network

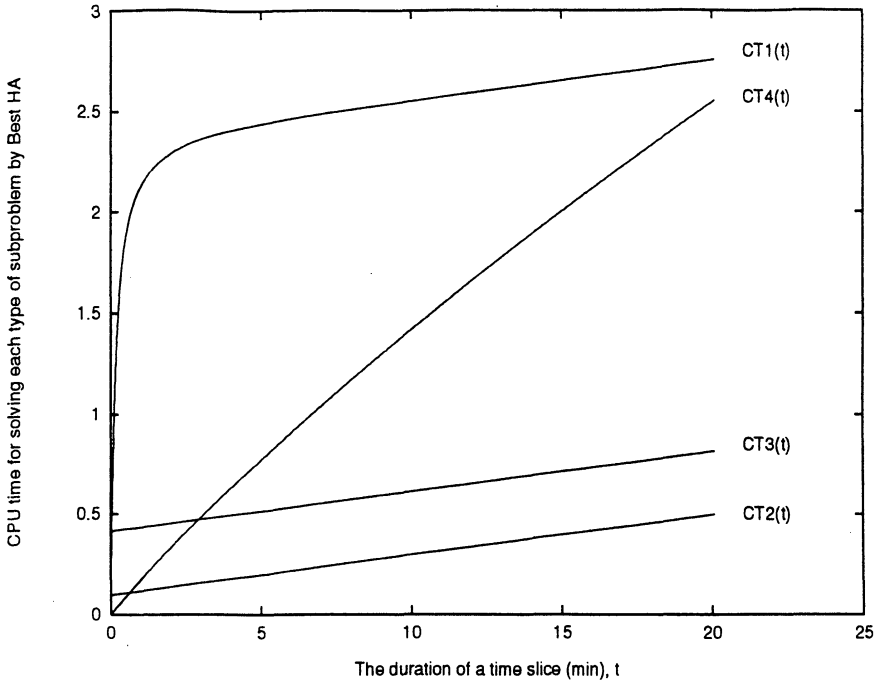

Figure 9: *Expected CPU times for solving the different subproblems in* Best HA.

and the micro-subnetworks so that an efficient implementation of *HA* can be achieved in the hierarchical network. As we have mentioned, the efficiency of *HA* strongly depends on the choice of decomposition. The size of the macro-network and micro-subnetworks affect the amount of computational effort required for solving subproblems of different types. For instance, consider the expected CPU times required by *Best HA* for solving subproblems of types 1–4. A large amount of CPU time required in Phase II (subproblems of type 4) suggests reducing the number of macronodes within each micro-subnetwork, thus reducing the size of the macro-network. On the other hand, a high computational cost of solving subproblems of type 2 or type 3 suggests reducing the sizes of micro-subnetwork, thus increasing the number of micro-subnetworks. These insights can be very helpful for constructing the network into hierarchical structure at the early stage of implementation.

6 Appendix

6.1 Proof for Theorem 3.2

In Phase I, the shortest paths that need to be solved by *Nearest HA* are the shortest paths for all O/D-pairs within every micro-subnetwork, and in the macro-network. The total number of operations required in this phase is thus

$$\Theta(N^k) \cdot C_D(\Theta(N^{1-k})) + C_D(\Theta(N^m))$$

which is

$$\mathcal{O}(N^{\max(2-k,2m)} \log N)$$

and

$$\Omega(N^{\max(2-k,2m)}).$$

In Phase II, the nearest macronode is selected from $\Theta(N^{m-k})$ surrounding macronodes for each micronode, requiring a total of $\Theta(N^{1+m-k})$ operations. Furthermore, to obtain lengths of the approximate shortest paths which are across subnetworks, one addition is needed for each path if the path connects a macronode and a micronode, while two additions are needed if the path connects two micronodes. It is easy to see that the number of paths of the second type is much more important, and that, independent of the values of m and k, there are $\Theta(N^2)$ such O/D-pairs. (There are $\Theta(N^{1-k})$ nodes per subnetwork, and $\Theta(N^k)$ subnetworks, yielding $\Theta\left(\left(N^{1-k}\right)^2 \cdot \left(N^k\right)^2\right)$ relevant O/D pairs.)

Since $k > 0$ and $m < 1$, the combination phase (Phase II) outweighs the shortest path phase (Phase I), so adding up the number of operations for both phases, we get

$$C_N(N) = \Theta(N^2).$$

The result now follows easily from the complexity results for Dijkstra's algorithm.

6.2 Proof for Theorem 3.3

The number of operations in Phase I is the same as for *Nearest HA*, i.e.

$$\mathcal{O}(N^{\max(2-k,2m)} \log N)$$

and

$$\Omega(N^{\max(2-k,2m)}).$$

In Phase II, the best approximating path is selected from $\Theta(N^{2(m-k)})$ possible paths (corresponding to that number of possible pairs of macronodes for the originating subnetwork and the destination subnetwork respectively). Since there are again $\Theta(N^2)$ relevant O/D-pairs, this phase requires a total of $\Theta(N^{2+2(m-k)})$ operations.

Again it is clear that the combination phase (Phase II) outweighs the shortest path phase (Phase I), so adding up the number of operations for both phases, we get

$$C_B(N) = \Theta(N^{2+2(m-k)}).$$

The result now follows easily from the complexity results for Dijkstra's algorithm.

6.3 Proof for Theorem 3.4

We will again distinguish between the number of operations required by *Nearest HA* in Phase I and Phase II.

In Phase I, several shortest paths have to be solved in various subnetworks. First, we solve for all shortest paths within a subnetwork from each origin, which requires a total of $\mathcal{O}(N^{\gamma+1-k}\log N)$ and $\Omega(N^{\gamma+1-k})$ operations. Next, the shortest paths for all pairs of nodes have to be computed in the macro-network, since we have assumed that there is at least one origin within each micro-subnetwork. This requires $\mathcal{O}(N^{2m}\log N)$ and $\Omega(N^{2m})$ operations. Finally, within all micro-subnetworks containing at least one destination, the shortest paths have to be computed from all macronodes contained in that micro-subnetwork. This requires $\mathcal{O}(N^{m+1-k}\log N)$ and $\Omega(N^{m+1-k})$ operations. Adding the numbers of operations required for solving these subproblems brings the number of operations required in Phase I to

$$\mathcal{O}(N^{\max(\max(\gamma,m)+1-k,2m)}\log N)$$

and

$$\Omega(N^{\max(\max(\gamma,m)+1-k,2m)}).$$

In Phase II, the nearest macronode is selected from $\Theta(N^{m-k})$ surrounding macronodes for each micronode which can be an origin (or destination), requiring a total of $\Theta(N^{\gamma+m-k})$ operations. Furthermore, to obtain lengths of the approximate shortest paths which are across subnetworks, one addition is needed for each path if the path connects a macronode and a micronode, while two additions are needed if the path connects two micronodes. It is easy to see that the number of paths of the second type is much more important, and that, independent of the values of m and k, there are $\Theta(N^{2\gamma})$ O/D pairs.

The total number of operations required by *Nearest HA* now satisfies

$$C_N(N;\gamma) = \mathcal{O}(\max(N^{\max(\max(\gamma,m)+1-k,2m)}\log N, N^{\gamma+\max(\gamma,m-k)}))$$

and

$$C_N(N;\gamma) = \Omega(N^{\max(\max(\gamma,m)+1-k,2m,2\gamma)}).$$

The result now follows in a straightforward manner from the complexity results for Dijkstra's algorithm.

6.4 Proof for Theorem 3.5

The number of operations in Phase I is the same as for *Nearest HA*, i.e.

$$\mathcal{O}(N^{\max(\max(\gamma,m)+1-k,2m)}\log N)$$

and

$$\Omega(N^{\max(\max(\gamma,m)+1-k,2m)}).$$

In Phase II, there are $\Theta(N^{2\gamma})$ O/D pairs which require combining paths obtained from Phase I, at a cost of $\Theta(N^{2(m-k)})$ per combination. Thus, $\Theta(N^{2(\gamma+m-k)})$ operations are necessary for Phase II. Thus, the total number of operations required by *Best HA* satisfies

$$C_B(N;\gamma) = \mathcal{O}(\max(N^{\max(\max(\gamma,m)+1-k,2m)} \log N, N^{2(\gamma+m-k)}))$$

and

$$C_B(N;\gamma) = \Omega(N^{\max(\max(\gamma,m)+1-k,2(\gamma+m-k))}).$$

The result now follow in a straightforward manner from the complexity results from Dijkstra's algorithm.

References

[1] A.V. Aho, J.E. Hopcroft, and J.D. Ullman, *Data Structures and Algorithms*, Addison-Wesley Publishing Company, 1983.

[2] J.C. Bean, J.R. Birge, and R.L. Smith, Aggregation in Dynamic Programming, *Operations Research*, vol.35, pp.215-220, 1987.

[3] A. Boneh and M. Hofri. The coupon-collector problem revisited, Technical Report CSD-TR-952, Computer Sciences Department, Purdue University, West Lafayette, Indiana, February 1990.

[4] R.J. Caron and J.F. McDonald. A new approach to the analysis of random methods for detecting necessary linear inequality constraints. *Mathematical Programming*, 43:97–102, 1989.

[5] E.V. Denardo. *Dynamic Programming: Models and Applications.* Prentice-Hall, Englewood Cliffs, New Jersey, 1982.

[6] S.E. Dreyfus, and A. M. Law, *The Art and Theory of Dynamic Programming*, Academic Press, New York, 1978.

[7] D.E. Kaufman, and R.L. Smith, Fastest Paths in Time-Dependent Networks for Intelligent Vehicle-Highway Systems Application, *IVHS Journal*, vol.1(1), pp.001-011, 1993.

[8] H.E. Romeijn, and R.L. Smith, Parallel Algorithms for Solving Aggregated Shortest Paths Problems, Technical Report 85-8, Department of Industrial and Operations Engineering, The University of Michigan, Ann Arbor, Michigan, 1993.

[9] *Survey of Regional Traffic Volume Patterns in Southeast Michigan*, SEMCOG Technical Report, 1985.

[10] H.S. Wilf, *Algorithms and Complexity*, Prentice-Hall, Inc., Englewood Cliffs, N.J., 1986.

[11] K.E. Wunderlich, and R.L. Smith, Large Scale Traffic Modeling for Route-Guidance Evaluation: A Case Study, IVHS Technical Report 92-08, The University of Michigan, Ann Arbor, Michigan, 1992.

[12] T. Yagyu, M. Fushimi, Y. Ueyama, and S. Azuma, Quick Route-Finding Algorithm, Technical Report, Toyota Motor Corp. and Matsushita Electric Industrial Co., Ltd., 1993.