An Efficient Shortest Path Computation System for Real Road Networks

Zhenyu Wang[1], Oscar Che[2], Lijuan Chen[2], and Andrew Lim[1,2]

[1] School of Computer Science and Engineering, South China Univ of Technology, Guangdong, China
[2] Dept of IELM, HKUST, Clear Water Bay, Kowloon, Hong Kong

Abstract. In this paper, we develop an efficient system to compute shortest paths in real road networks. An optimal shortest path algorithm is proposed based on a two-level hierarchical network structure. A pre-computation technique is used to improve the time efficiency. To further improve time efficiency and reduce memory requirement, we propose an algorithm to minimize the number of boundary nodes by relocating them between adjacent sub-networks. The performances of our approach with different network partition methods (with or without minimization of the number of boundary nodes) are compared in terms of both time efficiency and memory requirement. The experimental results on the real road network of Hong Kong demonstrated the efficiency of our method and the usefulness of minimizing of the number of boundary nodes.

Keywords: systems for real life applications, decision support, shortest path computation, hierarchical network structure.

1 Introduction

Efficient shortest path computations in real road networks are essential to *Intelligent Transportation System* (ITS) and other vehicle routing services. However, naive application of shortest path algorithms in real road networks always deteriorates because of the huge number of nodes. In the literature, various shortest path algorithms [13], [12] are available for route finding, of which the most popular one is Dijkstra's algorithm. Dijkstra's algorithm has a runtime complexity of $O(E + VlogV)$ for a network with E edges and V nodes. It performs well on a network with a small number of nodes. However, for a real road network with tens of thousands of nodes, the performance drops dramatically in term of time efficiency. Its application in the real road network thus requires some methods to reduce the number of nodes in consideration.

In the literature, a lot of work has been focused on the idea of hierarchical structures [4], [10], [3], [1]. The road network is organized into a hierarchical structure with the help of existing topographical knowledge, and heuristic methods are used to improve the time efficiency. Liu [10] partitioned the entire network into many small sub-networks by major roads. The shortest path algorithm was applied only to the sub-networks containing the origin or the destination and the

M. Ali and R. Dapoigny (Eds.): IEA/AIE 2006, LNAI 4031, pp. 711–720, 2006.
© Springer-Verlag Berlin Heidelberg 2006

major road network. Jagadeesh *et al.* [3] proposed a heuristic improvement by using Euclidean distances, instead of real distances, in the origin and destination sub-network to improve the time efficiency. However, with these methods the results are not guaranteed to be optimal. Researchers have also developed optimal models for shortest path computation based on hierarchical network structures [4], [6]. Pre-computation was proposed as well to improve the time efficiency of shortest path queries. But pre-computation leads to another problem, i.e., the memory requirement is tremendous [11]. Ning *et al.* [4] stored all-pair shortest paths of each fragment graph rather than the shortest paths of the whole network. This approach balances time efficiency and memory requirement and is highly complimentary.

In a hierarchical approach, the partition method is of great importance because it directly influences the time efficiency. Habbal *et al.* [2] proved that the most favorable decomposition schemes are those in which the number of sub-networks is relatively small, the sub-networks are of equal size, the number of boundary nodes per sub-network is uniform and the total number of boundary nodes is as small as possible. Ning *et al.* [4], [5] developed a link-sorting partition algorithm which is efficient for large map fragmentation, and proposed the *Spatial Partition Clustering* (SPC) technique. Karypis and Kumar [8] presented a multilevel algorithm for multi-constraint graph partitioning such that the partitions satisfy a balancing constraint while aiming at minimizing the edge-cut.

In this paper, we present an efficient system for shortest path computation and queries in a real road network which guarantees the optimality. In our model, the hierarchical structure is adopted to reduce the number of nodes in consideration so as to improve the time efficiency. The pre-computation technique is applied to facilitate shortest path queries. The road network is partitioned into a number of sub-networks, and the trade-off between time efficiency and memory requirement is balanced by only storing all-pair shortest paths of each sub-network. To further improve time efficiency and reduce memory requirement, we propose an algorithm to minimize the number of boundary nodes by relocating them between adjacent sub-networks. To evaluate the performance of our approach, computational experiments were conducted on the Hong Kong road network with 13626 nodes and 28735 edges. The edge-node ratio is 2.11.

2 Methodology

2.1 Partitioning Tools

To partition the real road network appropriately, we studied the SPC method, the multilevel partitioning algorithm and a computer-aided partition method with topographical knowledge. SPC is a link-sorting based partitioning method, exploiting unique properties of transportation networks such as spatial coordinates and high locality. The algorithm is based on the *plane-sweep* technique commonly used in multi-dimensional spatial data operations. For the details of the algorithm, refer to [5]. In our implementation, the nodes, instead of the links, are partitioned into clusters, and the nodes of the same cluster are grouped

together to form a sub-network. After all the nodes have been grouped into sub-networks, the entire network is successfully partitioned into a set of sub-networks and each node belongs to exactly a single sub-network. We also studied multi-level partitioning algorithms presented by Karypis and Kumar [8], [9], [7], and used the multilevel partitioning tool, METIS. METIS is a powerful software package for partitioning large graphs, partitioning large meshes, and computing fill-reducing orderings of sparse matrices.

The computer-aided partitioning method (CAP) is facilitated by human knowledge on the topographical respect of the real map. Through this approach, we divide the entire map into several sub-networks by manually selected boundary nodes. The selection of boundary nodes is subject to topographical knowledge of the map. When selecting the boundary nodes, we hope to isolate the regions with fewer roads connected with outside. For example, the Hong Kong Island is favorable to be isolated as a sub-map, which is connected with Kowloon through only 3 tunnels. So is the Lan Tau Island, which is linked with outside by the Tsing Ma Bridge. However, when it is difficult to further partition a dense sub-network, we have to select more boundary nodes. In such cases, it is necessary for us to study the specific region and then select as few boundary nodes as possible.

Moreover, a computer program was written to help us choosing the boundary nodes. The boundary nodes are selected one after another. After a boundary node is selected (or un-selected), the program displays the total number of sub-networks with current set of boundary nodes. For each sub-network, the program reports its size and the number of boundary nodes contained in it. This information is provided to help us deciding whether our previous choices of boundary nodes are good or not. With the help of the computer program and human knowledge, the Hong Kong road network was successfully partitioned into 74 components, each containing 52 to 309 nodes.

2.2 Optimal Shortest Path Computation Based on a Hierarchical Network Structure

We propose a shortest path algorithm based on a 2-level hierarchical network structure constructed by the SPC, METIS or computer-aided partitioning method. Our algorithm guarantees the optimality of the results. In order to explain the algorithm, we describe first the 2-level hierarchical network structure. After partitioning, the original network $G(V,E)$ is divided into sub-networks: SG_1, SG_2, ..., SG_n, called the *low-level* networks. In each sub-network, the nodes with an outgoing edge to or an incoming edge from other sub-networks is defined as *boundary nodes*. For example, consider Figure 1 where a digraph G and its sub-networks SG_1, SG_2 and SG_3 are shown. The boundary node set of SG_1, SG_2 and SG_3 is {E, F}, {G, H, J, L} and {M, N, O}, respectively.

We define the local shortest path between two boundary nodes computed within each sub-network as $SP_i(boundary_1, boundary_2, distance)$, where i corresponds to SG_i, the sub-network which the two boundary nodes belong to. Firstly, we apply Dijkstra's algorithm in each *low-level* network to compute all $SP_i(boundary_1, boundary_2, distance)$'s and store them. In Figure 1, for example,

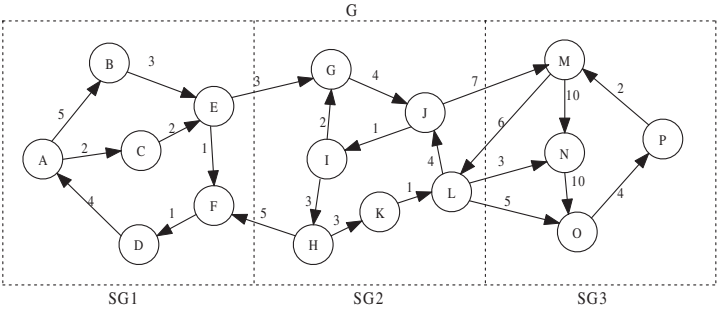

Fig. 1. A digraph G and its subgraph SG_1, SG_2 and SG_3

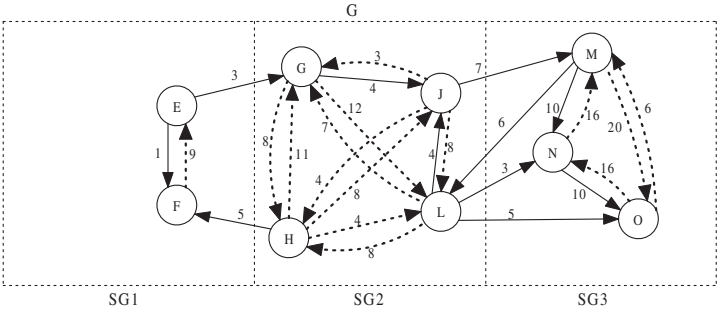

Fig. 2. The high-level network constructed from SG_1, SG_2 and SG_3

the local shortest path from boundary node E to F, computed within SG_1, is $SP_1(E, F, 1)$, the shortest path from G to H in SG_2 is $SP_2(G, H, 8)$, from M to N in SG_3 is $SP_3(M, N, 10)$, and so on.

We isolate all the boundary nodes, each two of which within the same sub-network are linked with two additional edges equivalent to the local shortest paths between them. Then add the outgoing or incoming edges to the corresponding boundary nodes. Thus we construct the *high-level* network, which is the connector of all the sub-networks. The high-level network together with the low-level sub-networks forms the 2-level network structure. The high-level network constructed from SG_1, SG_2 and SG_3 of Figure 1 is given in Figure 2, where the newly-added edges are represented by dashed arrows.

Apply the all-pair shortest path algorithm (that is, apply Dijkstra's algorithm repetitively) in the high-level network, and all-pair global shortest paths of the high-level network are determined, including the global shortest paths between each pair of boundary nodes that belong to the same sub-network. The shortest path between two boundary nodes computed in the high-level network is denoted as $SP_h(boundary_1, boundary_2, distance)$. The results are stored in memory. For example, the distance from M to N in the high-level network is $SP_h(M, N, 9)$, by going though L of SG_2.

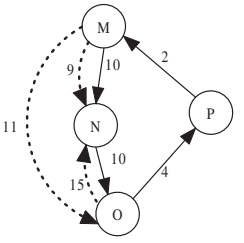

Fig. 3. The updated SG_3 with newly-added edges

Then we return to the low-level networks. Each pair of boundary nodes in the same sub-network SG_i are linked with an additional edge of weight $SP_h(boundary_1, boundary_2, distance)$, if $SP_h(boundary_1, boundary_2, distance)$ is less than $SP_i(boundary_1, boundary_2, distance)$. For example, the updated SG_3 is given in Figure 3, where the newly-added edges are represented by dashed arrows. After that we apply all-pair shortest path algorithm in each updated sub-network to compute global shortest paths between any two nodes of the sub-network. The results are then stored in memory. At the point, the global all-pair shortest paths for the high-level network and for each sub-network have been computed and stored. Thus, the all-pair shortest path initialization process is completed. Since there is a trade-off between time efficiency and memory requirement, we do not store all-pair shortest paths of the whole road network in our system. Instead, we only store all-pair shortest paths in each low-level network and the high-level network, which saves memory significantly without greatly compromising the time efficiency.

To retrieve the shortest path between two given nodes, s (the origin) and t (the destination), we should first check whether they are boundary nodes and where they are located. In the following, we denote the global shortest path from the origin to the destination as $SP(s, t, distance)$, and the set of boundary nodes of a sub-network SG_i as BN_i. There are 4 scenarios listed as follows.

Case 1: If the two nodes are both boundary nodes, $SP(s, t, distance) = SP_h(s, t, distance)$. For this case, retrieve the result from the high-level network.

Case 2: If the two nodes are located in the same low-level network, SG_i, $SP(s, t, distance)$ has already been computed and stored in SG_i. For this case, retrieve the result from SG_i.

Case 3: If s is inside a low-level network SG_i and t is a boundary node, we create a directed graph $G' = (V', E')$, where V' contains s, BN_i and t. For each boundary node, v, in BN_i, there is a directed edge in E' from s to v with weight $SP(s, v, distance)$, and another directed edge from v to t with weight $SP(v, t, distance)$. Then we apply Dijkstra's algorithm to obtain the shortest path from s to t in G'. Vise versa for the case that s is a boundary node but t is not. The auxiliary graph G' for computing $SP(P, L, distance)$ is given in Figure 4, from which we get $SP(P, L, 8)$.

Case 4: If s is inside a low-level network SG_i and t is inside a different low-level network SG_j , we create a directed graph $G' = (V', E')$, where V' contains

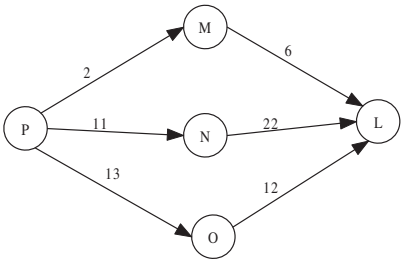

Fig. 4. The auxiliary graph G' for $SP(P, L, distance)$

s, BN_i, BN_j and t. For each boundary node, v, in BN_i, there is a directed edge in E' from s to v with weight $SP(s, v, distance)$. For each boundary node, v', in BN_j, there is a directed edge in E' from v' to t with weight $SP(v', t, distance)$. In addition, there is a directed edge from v to v' with weight $SP(v, v', distance)$. Then we apply Dijkstra's algorithm to obtain the shortest path from s to t in G'.

It is straight-forward to extend the above retrieval algorithm to retrieve the shortest paths from s to a set of destination nodes, $\{t_1, t_2, \ldots, t_n\}$. Thus, the description is omitted here.

2.3 Minimization of the Number of Boundary Nodes

From the algorithm described in the previous section, we realized that the number of boundary nodes affects both time efficiency and memory requirement. Since the boundary nodes are the entry/exit points of the low-level networks, the fewer they are, the fewer entry/exit points we need to consider in Case 3 and 4 of the retrieval algorithm. Moreover, in our implementation we precompute and store all-pair shortest paths of the high-level network, of which the nodes are exactly the boundary nodes. With fewer boundary nodes, less memory is required for the high-level network. Therefore, we took minimization of the number of boundary nodes into consideration and developed an algorithm for it. The algorithm improves the partitions generated by the methods as described in Section 2.1 by relocating the boundary nodes between different sub-networks.

Refer to Figure 1. The boundary node set of SG_1, SG_2 and SG_3 is {E, F}, {G, H, J, L} and {M, N, O}, respectively. If we move a boundary node from its native sub-network to another sub-network to which it is adjacent, we may reduce the number of boundary nodes. For example, if we move the boundary node L from SG_2 to SG_3, the resulting boundary node set for SG_2 and SG_3 is {G, H, J, K} and {L, M}, respectively. The number of boundary nodes is then reduced by 1.

To interpret the minimization algorithm formally, we define the *adjacent nodes* of a boundary node, v, as the nodes which are directly linked with v by an incoming or outgoing edge. An *adjacent sub-network* of v is a non-native sub-network which contains at least one of its adjacent nodes. In the native sub-network of v, there are two types of adjacent nodes: those adjacent nodes which are boundary nodes themselves (denoted as *native-boundary-nodes*) and those which are not

(*native-regular-nodes*). In an adjacent sub-network of v, there are also two types of adjacent nodes: those linked with no other foreign boundary node except v (*single-linked-nodes*) and those linked with other foreign boundary nodes (*multi-linked-nodes*). If v is moved to an adjacent sub-network, the *native-boundary-nodes* will remain as boundary nodes, while the *native-regular-nodes* will become new boundary nodes. On the other hand, the *single-linked-nodes* in the target adjacent sub-network will become non-boundary nodes, while the *multi-linked-nodes* will remain as boundary nodes. v will become a non-boundary node only if it has only one adjacent sub-network and does not have any *native-boundary-node* or *native-regular-node*. Our minimization algorithm keeps on moving boundary nodes to their adjacent sub-networks as long as the total number of boundary nodes is reduced. It terminates at a local optimum where no further reduction is possible.

3 Computational Results

3.1 Comparison of Different Partitioning Methods

The partitioning methods (SPC, METIS and CAP) were tested on the Hong Kong road network. Figure 5 and 6 illustrate the all-pair shortest path computation time and memory usage of METIS, as the number of partitions, n_p, increases. The corresponding figures for SPC are omitted due to its poor performance. The best results for METIS, SPC and CAP are given in Table 1, where "+M" indicates that the partition method is enhanced by minimizing the number of boundary nodes.

The results indicate that the best value of n_p is about 90 for METIS and 50 for SPC. Table 1 clearly shows that METIS performed much better than SPC, while CAP is better than METIS. The memory requirement of CAP+M is about 4.82% less than METIS+M. More importantly, the all-pair shortest path computing time of CAP+M is over 1 second (28.62%) less than METIS+M, demonstrating the usefulness of topographic knowledge. Moreover, the results with minimization of the number of boundary nodes are always better than those without minimization, which demonstrates the strength of the minimization algorithm.

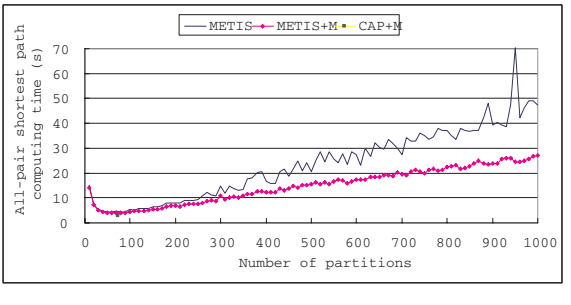

Fig. 5. All-pair shortest path computation time with METIS

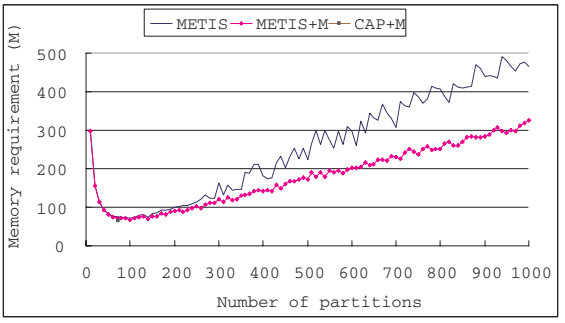

Fig. 6. Memory requirement with METIS

Table 1. The best results with METIS, SPC and CAP

Partition scheme	n_p	All-pair time(s)	n_p	Memory usage(M)
METIS	90	4.234	90	73.158
METIS+M	90	3.875	100	67.637
SPC	30	10.953	50	132.548
SPC+M	50	6.609	50	96.941
CAP	74	3.000	74	65.861
CAP+M	74	2.766	74	64.378

3.2 Time Comparison for One-One Shortest Path Computation

For one-one shortest path computation, our approach based on METIS+M ($n_p = 90$) and CAP+M was compared with direct application of Dijkstra's algorithm. Figure 7 gives their average distance retrieval time, with the number of queries ranging from 1 to 100. The computation time of our approach is at the level of 10^{-3}s, much less than that of Dijkstra's algorithm, which indicates the superiority of our approach. In addition, the distance time of CAP+M is less than that of METIS+M, indicating the advantage of topographic knowledge in partitions. Similarly, for the path retrieval time, Dijkstra's algorithm was much worse than our approach, and CAP+M out-performed METIS+M again.

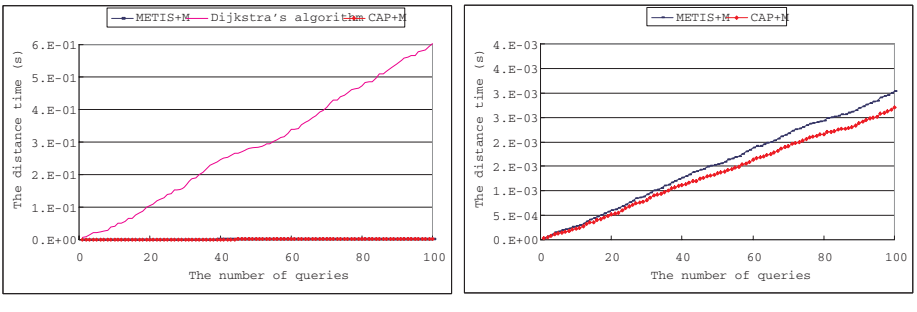

Fig. 7. Comparison of the path time

3.3 Time Comparison for Some-Some and All-All Shortest Path Computations

For some-some shortest path computation, we set the node pairs as 100-100, 200-200, 400-400, 600-600, 800-800, 1000-1000, 2000-2000, 4000-4000, 6000-6000, 8000-8000 and 10000-10000. The results of our approach with METIS+M and CAP+M are compared with those of the Dijkstra's algorithm in Table 2. The all-all results are given in Table 3. From Table 2 and 3, it is clear that the distance time of the Dijkstra's algorithm is more than that of our approach with METIS+M or CAP+M. Again, the distance time with CAP+M is less than METIS+M.

Further analysis of Table 2 and 3 reveals that, for n-n distance queries, the superiority of our approach decreases with the increasing of n. For example, the ratio of CAP+M time to the Dijkstra time increases from 3.76% to 65.90%. The underlining rationale is that each run of Dijkstra's algorithm always computes the shortest paths from a source node to all other nodes. Each *1-n* query takes nearly the same time, regardless of the value of n. When n is small, most of the one-all shortest paths computed by the Dijkstra's algorithm are un-used and our approach with pre-computation is thus much more efficient. As n increases, the n-n Dijkstra time increases linearly with regard to n. For each *1-n* query, our approach requires n *1-1* operations, if without optimization. By grouping queries with destination nodes in the same sub-network, we are able to improve our *1-n* query processing. Even so, our retrieval time still increases much faster than the Dijkstra time. This explains why the time saving of pre-computation gradually decreases as n increases.

Table 2. Computation performance comparison for some-some shortest paths

Some-some	METIS+M(s)	CAP+M(s)	Dijkstra's algorithm(s)
100-100	0.093	0.047	1.250
200-200	0.235	0.187	2.500
400-400	0.734	0.500	4.969
600-600	1.281	0.922	7.453
800-800	1.891	1.406	10.000
1000-1000	2.578	1.969	12.406
2000-2000	6.969	5.453	24.781
4000-4000	19.969	15.797	49.031
6000-6000	38.328	30.453	73.532
8000-8000	62.375	49.828	98.094
10000-10000	92.078	72.843	122.485

Table 3. Computation performance comparison for all-all shortest paths

All-all	METIS+M(s)	CAP+M(s)	Dijkstra's algorithm(s)
13626-13626	137.14	110.813	168.141

4 Conclusion

In this paper, we developed an optimal shortest path algorithm, based on a two-level hierarchical network structure, to compute shortest paths in real road

networks. Pre-computation was used to improve the time efficiency. We also proposed an algorithm to minimize the number of boundary nodes by relocating them between adjacent sub-networks. The experimental results on Hong Kong road network demonstrated the efficiency of our method and the usefulness of minimizing of the number of boundary nodes.

References

1. Ashok K. Goel, Todd J. Callantine, Murali Shankar, and B. Chandrasekaran. Representation, organization, and use of topographic models of physical spaces for route planning. In *Proc. the 7th IEEE Conference on Artificial Intelligence Applications*, pages 308–314. IEEE Computer Society Press, 1991. Miami Beach, Florida, February 1991.
2. Mayiz B. Habbal, Haris N. Koutsopoulos, and Steven R. Lerman. A decomposition algorithm for the all-pairs shortest path problem on massively parallel computer architectures. *Transportation Science*, 28(4):292–308, 1994.
3. G. R. Jagadeesh, T. Srikanthan, and K. H. Quek. Heuristic techniques for accelerating hierarchical routing on road networks. *IEEE Transactions on Intelligent Transportation Systems*, 3(4):301–309, 2002.
4. Ning Jing, Yun-Wu Huang, and Elke A. Rundensteiner. Hierarchical encoded path views for path query processing: An optimal model and its performance evaluation. *IEEE Transactions on Knowledge and Data Engineering*, 10(3):409–432, 1998.
5. Ning Jing, Yun-Wu Huang, and Elke A. Rundensteiner. Optimizing path query performance: Graph clustering strategies. *Transportation Research Part C*, 8(1-6):381–408, 2000.
6. Sungwon Jung and Sakti Pramanik. An efficient path computation model for hierarchically structured topographical road maps. *IEEE Transactions on Knowledge and Data Engineering*, 14(5):1029–1046, 2002.
7. George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *University of Minnesota, Department of Computer Science/ Army HPC Research Center, Minneapolis, MN 55455 Technical Report No.95-035*, 1998.
8. George Karypis and Vipin Kumar. Multilevel algorithm for multi- constraint graph partitioning. *University of Minnesota, Department of Computer Science/ Army HPC Research Center, Minneapolis, MN 55455 Technical Report No.98-019*, 1998.
9. George Karypis and Vipin Kumar. Multilevel k-way partitioning scheme for irregular graphs. *University of Minnesota, Department of Computer Science/ Army HPC Research Center, Minneapolis, MN 55455 Technical Report No.95-064*, 1998.
10. Bing Liu. Route finding by using knowledge about the road network. *IEEE Transactions On Systems, Man, and Cybernetics-Part A: Systems and Humans*, 27(4):436–448, 1997.
11. Shashi Shekhar, Andrew Fetterer, and Brajesh Goyal. Materialization trade-offs in hierarchical shortest path algorithms. In *Proc. the 5th International Symposium on Advances in Spatial Databases*, pages 94–111, 1997.
12. F. Benjamin Zhan. Three fastest shortest path algorithms on real road networks: Data structures and procedures. *Journal of Geographic Information and Decision Analysis*, 1(1):69–82, 1997.
13. F. Benjamin Zhan and Charles E. Noon. Shortest path algorithms: An evaluation using real road networks. *Transportation Science*, 32(1):65–73, February 1998.