Road Network Distance Based Efficient Algorithms Suitable in Location Based Services

Aye Thida Hlaing

A Thesis Submitted to the
Graduate School of Science and Engineering, Saitama University
in Partial Fulfillment of the Requirements for the Degree of
DOCTOR OF ENGINEERING
in
Mathematics, Electronics and Informatics

Supervisor: Professor Yutaka OHSAWA

Saitama University

2015, March

Abstract

This thesis describes the studying of efficient algorithms for real-time monitoring, shortest path finding and Reverse k nearest neighbor (R-kNN) query applying on road network distances for Location Based Services (LBS). In recent time, finding shortest path in real road networks is crucial for many applications including location-based services and mobile computing. Since mobility is constantly increasing, we can observe that also an increase in the time dependency of spatial information related to human activities. The role of location-based services is rising as the increasing numbers of users are requesting the location-based information. The main idea of LBS is to provide the service that depends on the positional information which associated with the user, most importantly, the user's current location. The service may also be dependent on other information, such as personal preferences and interests of the user. For example, finding the cheapest hotel within 20 km, where is the nearest gas station, how long it will take to go to Italy restaurant, are some specific user's queries in location-based services. Also a service may inform its users about traffic jams, weather situation, the position of emergency vehicles, hazardous materials or public transportation. Moreover, in other related activities like parcel management, tourism planning, bus routes queries for Urban planning, checking movement of terrorists in emergency situations for crime prevention, etc., require LBS services based on spatial information. For all these requirements, studying on some typical queries like nearest neighbor queries, range queries, spatial join queries, and reverse nearest neighbor queries has been demanded.

Various route queries in road networks have gained significant research interests due to advances in GIS and mobile computing such as car navigation system. The main challenge of processing such a query is how to efficiently monitor the moving object and how to retrieve the route rapidly. In recent time, much work has been conducted on a real time monitoring of moving objects (cars and humans). Some studies adopted to get high accuracy tracking of moving objects with less communication between moving objects and server. However, their studies assumed the moving object is moving in Euclidian space or in fixed route. This thesis proposes a real time monitoring method aiming for both thick client and thin client. We will discuss these two kind of clients detail later. Using the“ frequently used routes ”(FUR) information, we offer high scalability monitoring system with empirical comparisons of the proposed method and the conventional dead-reckoning on road network method.

Despite the importance of spatial networks in real-life applications, most of the spatial query methods focus on Euclidean distance, where the distance between two objects is determined solely by their relative position in space. However, in practice, objects can usually move only on a pre-defined set of routes as specified by the real road network (road, railway, river etc.). Thus, the important measure is the network distance, i.e., the length of the shortest trajectory connecting two objects, rather than their Euclidean distance. Only using Euclidean distance on spatial network databases (SNDB) is scarce and too restrictive for emerging applications such as mobile computing and location-based queries.

The main difference between using Euclidean distance and road-network distance is based on their calculation costs. As considering, the Euclidean distance between two arbitrary points can be computed easily, however, in the road-network distance, it takes longer processing time. For example, if the river or mountain lies between two points, there is totally difference between the Euclidean distance and the road-network distance, and the costs of the distance calculation as well. In the conventional methods, to promptly acquire the distance between two termi-

nal points for spatial queries in LBS applications, we can use well-known shortest path finding algorithms, Dijkstra's algorithm and A* algorithm. However, due to the complexity of some queries, for example, ANN (aggregate nearest neighbor) queries, RNN (reverse nearest neighbor) queries, skyline queries, k-NN queries and several kind of TPR (trip planning route) queries, needed efficient algorithm to find the shortest paths between starting point and target point.

Dijkstra's algorithm and A* algorithm can be directly applied to these queries, nevertheless these two algorithms take very long processing time and that will cause high calculation cost. Therefore, we propose another efficient shortest path finding algorithm, based on materialized-path-view constructed only on partitioned subgraphs to compute this road network distance.

As a shortest path query is a basic operation in several types of queries, efficient path computation is essential in such kind of queries, for example, k-NN queries, ANN queries, R-kNN queries and trip planning queries. In existing path finding algorithms, Dijkstra's and A* algorithms, require pre-computed all-pair shortest paths and store them on-line. A* algorithm can reduce the path computation but it is not efficient for the re-computation or update of the flat path view for large networks.

Other Materialize Path View (MPV) approaches also have been proposed for a shortest path query based on the pre-computed distance table. This method needs $O(n^2)$ space when the given number of the nodes is n. For this reason, it is not suitable for the large network as well. Distance materialization methods based on types of roads have been used also in Japan car navigation system. There are several types of roads according to the road classification in Japan, for example, highways or express ways, national roads and residential roads etc. In most existing studies, their methods are suitable for searching long distance trip path and mostly aiming for using express ways. However, in location based services applications, there are requested to search shortest path finding in small area and points of interest (POIs)

are normally existed on residential roads, and not on highways or express ways. Therefore, we propose a fast shortest path query strategy suitable for LBS as in small search area and trip is also considered for usual roads.

Then, several hierarchical representation methods have been proposed to reduce the amount of data. In HEPV (hierarchical encoded path view) and HiTi graph methods, using hierarchical representation and semi-materialized approach have been proposed to reduce the large amount of data. In this thesis, we propose the shortest path finder with light materialized path view. Our study is closely related with their works, however, their searching assumes the hierarchical structure essentially and we will discuss this difference detail in other session.

Using partitioned subgraphs, we study one more efficient algorithm for Reverse k-Nearest Neighbors (R-kNN) query on road network distance. According to our knowledge, the existing methods for R-kNN queries required to find kNN search on every visited node. This causes a large number of node expansions and the processing time is increase simultaneously. Therefore, we propose a fast R-kNN search algorithm that running based on a simple materialized path view (SMPV) structure and we adopted incremental Euclidean restriction strategy for kNN queries which is the main function in R-kNN queries.

Contents

CHAPTER 1

Introduction

Over the last few decades, spatial databases and mapping services have become one of the most important applications in queries services based on geographical positioning system. With the wide usage of location tracking systems, tracking relationships among moving objects over their location changes is possible and important to many real applications. The prevalence of Global-Positioning-Devices (GPS) enables to track and coordinate large numbers of continuously moving objects, and store their positions in databases. This results in new challenges for query optimization in such databases because it is no longer sufficient to have a query processing strategy that is tailored to meet the problems in data storage and processing cost, may be travelling time or travelling distance. Rather, query processing in mobile databases needs to consider the problems specific to the behavior of mobile devices such as network traffic, etc. Particularly, the main goal in many applications is to minimize the processing cost of retrieving the shortest path and, as a consequence, to minimize the data storage . In existing studies, two well known shortest path finding algorithms are Dijkstra's algorithm and A* algorithm. In addition, Incremental Euclidean Restriction (IER) strategy[1] has been applied in queries algorithm based on road network distance for LBS applications.

1.1 Some Types of Spatial Queries in Location Based Services (LBS)

We introduce some kinds of queries for the purpose of well understanding the characteristic of spatial queries in location based services. Depending on the purpose of users' preferences, several types of queries can be mainly introduced into spatial network queries according their requirements. Mostly in LBS, destination points of a query point(POIs), for example, hotels, banks and grocery stores etc., have been searched. Especially, POIs which can fulfill the searching requirements are resulted. Examples of these queries in LBS are described here, k nearest neighbor (k-NN) queries, reverse nearest neighbor (RNN) queries, aggregate nearest neighbor (ANN) queries, and trip planning queries.

In k nearest neighbor (k-NN) query, given a query point q and a set of POIs P, a k-NN queries are searching for k number of nearest POIs in a specific area. For example, if someone wants to find one of the nearest gas station, at this situation, kNN query can find starting from one nearest gas station to k^{th} nearest gas station based on the current location area according to the distances (i.e., Euclidean distance or network distance) or travelling time.

For Reverse nearest neighbor (RNN)query, given a query object q, reverse nearest neighbor search finds all objects in the network whose nearest neighbors are the query object. Note that since the relation of NN is not symmetric, the NN of a query object might differ from its RNN(s). For example, RNN query can find the set of customers affected by the opening of a new shopping mall location in order to inform the relevant customers. This query can also be used to identify the location which maximizes the number of potential customers. Consider another example, an RNN query may be issued to find the existing shopping malls that are affected by opening a new shopping mall at some specific locations.

Aggregate nearest neighbor (ANN)query, is a query to satisfy the request of the

multiple query points (multiple users) who want to find the shortest routes based on some aggregate functions (e.g.,*sum*, *min* and *max*) related with travelling times or distances for all users. For example, Four peoples from different locations desire to find a cafeteria to meet. By applying a suitable aggregate function, their desire cafeteria is retrieved as a result which can optimize the total minimum travelling cost (i.e., distances or time) for four peoples.

In Trip planning query (TPQ), given the starting point, final destination points and POI categories to visit during the trip, TPQ retrieves the shortest route from the starting point to the destination point by visiting one POI from each different POI categories during the trip. In TPQ, the POIs visiting order is not specified. Due to this complexity, TPQ is difficult to solve. For example, one tourist starts his trip from airport and to be visited bank, restaurant and convenience store without specifying visiting order. Then he ends his trip at the hotel. Therefore, TPQ retrieves the shortest path starting from airport, passing through one bank, one restaurant and one convenience store in non-specifying order, and terminating at the hotel.

Shortest path finding is the important function in LBS. Dijkstra's algorithm and A* algorithm are well-known algorithm for shortest path finding in spatial queries. However, these algorithms are not suitable for very large spatial database and some complicated queries. Therefore, in this study, we propose Simple Materialized Path View (SMPV) structure for shortest path finding and Incremental Euclidean Restriction (IER) adaption.

1.2 Incremental Euclidean Restriction Framework of LBS

The Incremental Euclidean Restriction (IER) framework is used widely in several types of queries on road network. IER framework is mainly based on the concept

that the Euclidean distance between two terminal points is always a lower bound of the road network distance. In IER framework, firstly, it searches a set of k-NN points on the Euclidean distance using the R-tree [2], and then to confirm that points are truly k-NN points on the road network distance. In k-NN search on the Euclidean distance are not always same in k-NNs on the road network distance. As mention in earlier, k Nearest neighbor query retrieves the k-POIs closest to the query point.

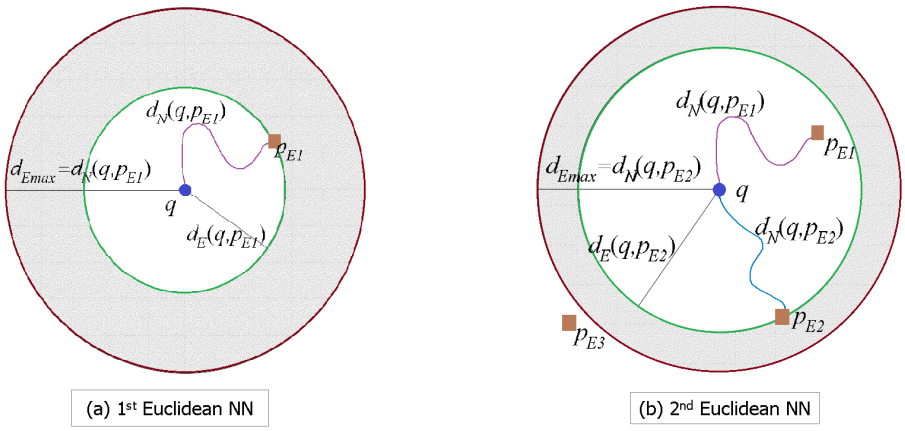


Figure 1.1: Finding the NN in IER [1]

In Fig. 1.1(a), specifically, assuming that only one NN is required, IER first retrieves the Euclidean nearest neighbor P_{E1} of q, using an incremental k-NN algorithm on the entity R-tree. Then, it requires to calculate the network distance between P_{E1} and q. Following the Euclidean lower bound restriction, other POIs closer to q than P_{E1}, should be within Euclidean distance $d_{Emax} = d_N(q, P_{E1})$ from q, i.e., they should exist in the shaded area of Fig. 1.1(a). In Fig. 1.1(b), the second Euclidean NN P_{E2} has been retrieved due to exist within the d_{Emax} range. If $d_N(q, P_{E2}) < d_N(q, P_{E1})$, P_{E2} will become the current NN and d_{Emax} will be up-

dated as $d_N(q, P_{E2})$, after the searching area becomes smaller (the shaded area in Fig. 1.1(b). Then the next Euclidean NN P_{E3} does not exist in searching area, the algorithm terminates with P_{E2} as the final result.

Finding k^{th} nearest neighbors can be performed by the following steps. In the first step, the k Euclidean NNs are obtained using Rtree, sorted in ascending order of their network distances to q, and d_{Emax} is set. In second step, similar to the 1NN case, the subsequent nearest neighbors in Euclidean distance are retrieved incrementally, until the next Euclidean NN has larger Euclidean distance than d_{Emax}. We apply this IER strategy to our proposed methods. The advantages of IER adaption is shown by the experiments in other session.

1.3 Contributions

In this thesis, we study a real time monitoring of moving objects based on their frequently used routes. For this study, we propose an efficient tracking algorithm for the positions of moving objects. Moreover, we apply an accurate map matching algorithm in shortest route retrieving algorithm. Our experimental results show how our proposed methods outperformed the existing dead-reckoning on road network method in both thick client and thin client.

After that, in our study, we focused on light material path view strategy to retrieve the shortest path hierarchically. We present SPF algorithm on three kinds of methods based on three variations referring different levels of distance materialization. The main difference between three variations of the SPF algorithm is the materialization level of the distance in the subgraphs. The performance of our proposed method is evaluated by the experiments compared with the conventional A* algorithm and the HEPV method.

And then, we proposed an adequate R-kNN queries algorithm based on that material path view method. Here, we employed IER strategy which used a filtering mechanism to rapidly generate a set of candidate POIs based on their Euclidean

distance from a query point. Then, the Euclidean distance for each candidate is verified by computing their road network distance. This IER adaption has been applied in k-NN query which is the main function in R-kNN query. For R-kNN query on road network distance required in LBS applications, we compare our proposed methods with the existing works to prove the efficiency and stability of the proposed algorithms.

1.4 Experimental Environments

In this thesis, we implement the experiments for all proposed methods comparing with the existing conventional methods. In all experiments, we use a real road network data. For the experiments, a digitized road map from a 1/25,000 scale base map that covers Saitama City, Japan is used. Points of interest (POIs) are randomly generated by a pseudo-random sequence. The probability value indicates the POIs density on road network map. For example, $Prob = 10^{-2}$ mean one POI exists on 100 road segments. And the processing time for both existing methods and proposed methods are shown in seconds (s).

1.5 Outline

Our main topic is efficient algorithms suitable in LBS based on road network distances, and some knowledge is described in Chapter (1). In Chapter (2), we discuss some related works. Proposed algorithms for real time monitoring of moving objects are presented in Chapter (3). In Chapter (4), shortest path finding algorithm with materialized path view is expressed. Applied SMPV structure in R-kNN queries which is suitable for LBS applications, has been discussed in Chapter (5). The conclusion of our thesis and the future research are contributed in the last Chapter (6).

CHAPTER 2

Related Work

Shortest path searching on a road network is essential for emerging Location Based Services (LBS) and many real time Geographic Information Systems (GIS) [[3][4][5][6][7]]. Location Based Service (LBS) is an information services which used in a variety of applications, especially to verify the location of persons or moving objects, such as to find the nearest restaurants, movie theatre, gas station or to find where is a friend. Due to the development of wireless communication and positioning technologies, in LBS, it is also require to make the improvements to the efficient path finding algorithms. In this chapter, we present the related work, background knowledge of our thesis and introduce well-known shortest path finding algorithms, query access indexing method and materialized path view method.

2.1 Shortest Path Finding Algorithms

A road network can be represented in weighted graph as $G = V, E$, where V is a set of nodes (that the point which is branching roads or merging roads on actual road map), and E is a set of edges between two nodes(that the road links between two network nodes). And attached value on each edge indicates the weight that

the distance or time travelling along that edge. For example, a person who wants to search the fastest route from one place to another using a road map, we defined vertices as locations and edges as the road segments. Then we calculated the weights of the routes by the time or distance needed to satisfy the user's requirement. This is a fundamental processing in navigation system and spatial queries based on the road network distance.

Fast shortest path search between two points on a road network is essential demand in Location Based Services (LBS). The shortest path finding method can be categorized into compute-on-demand method using adjacency list and precomputation method using road network distance materialization approaches. Among several different algorithms, Dijkstra 's algorithm [[8][9][10]] and A* algorithm [11] are well-known algorithms for first method.

Some general terms which are frequently used in the explanation for the queries on road network are described here. As mention in earlier, the road network can be represent in a weighted graph $G = E, V$ in which V is a set of nodes and E is a set of edges between these nodes. Each edge has a weight described the distance or time to travel along that edge. An edge can be a straight line or a polyline. If two nodes are directly connected by an edge, these two nodes are defined as adjacent nodes to each other.

Dijkstra 's algorithm and A* algorithm are well-known algorithms to search the shortest path from a query point to the target point. Although both algorithms refer an adjacency list of nodes to find the neighboring nodes to a currently noticed node, A* algorithm is faster than Dijkstra‘ s algorithm. In Dijkstra's algorithm, the search are is expended as the wave front in all directions. In A* algorithm, it uses a heuristic function and only focuses to reach the target node, so as not to reach every other nodes. Therefore, it can reduce the search area and unnecessary node expansions comparing with Dijkstra's algorithm.

2.1.1 Dijkstra's Algorithm

Dijkstra 's algorithm is also known as the single source shortest path problem. It can compute the shortest path from the source to each of other vertices in directed weighted graph and find the shortest path to the target node as well.

Dijkstra 's algorithm is performed by the following steps:

Initial state: Assign a distance value to every node i.e., set it to zero for initial node and to infinity for all other nodes

Step 1. Mark all nodes as unvisited. Set the initial node as a current node.

Step 2. For the current node, consider all of its unvisited neighboring nodes and calculate their distances. And compare that distances and choose one has minimum value. Then selected node is marked as visited and never be checked again.

Step 3. If the target node has been marked visited or if the minimum distance is infinity (if there is no connection between the initial node and remaining unvisited nodes), then search is stop and the algorithm is terminated.

Step 4. Iteration of the algorithm by selecting the unvisited node that is marked with minimum tentative distance, and set it as new current node then go back to step 2.

The example will briefly explain each step that is taken and how to find the shortest path in Dijkstra's algorithm using Fig. 2.1.

In Fig. 2.1, there are 5 vertices (node A, B, C, D and E). The value between the two vertices is defined as the weight of that two nodes. Using this figure, Dijkstra's algorithm will determine the shortest path from starting point (node A as shown in green color) to target point (node E as shown in red color).

In initial state, node A is set as an initial node and assign distance value to all nodes in figure, i.e., 0 for node A and ∞ for other nodes. For step 1, node A is set

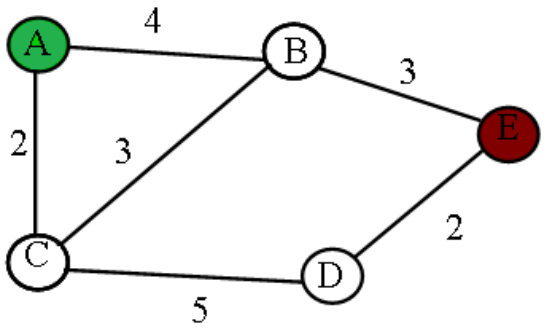


Figure 2.1: Weighted graph example in Dijkstra's Algorithm

as current node and other nodes are marked as unvisited. In step 2, checking the adjacent nodes of node A (node B and C), then the distances of adjacent nodes of node A are computed. The distance between A and B is 4 and the distance between A and C is 2. Then select node C and mark as visited due to minimum value. In step 3, the target node has not been examined yet. Therefore go to step 4, set up the node C as current node and go back to step 2. By iteration, node C is a current node and it is computed the distances of adjacent nodes of node C (now node B and node D) from source node A. Then compute their distances, the distance between A to B via C is 5 and the distance between A to D via C is 7. However, the direct link distance between A and B is only 4. Therefore selecting the direct link between A and B comparing other links via C and set node B as visited. Same iteration of node expansion at D, the distance between A to E via B is 7 and the distance between A to E via D is 9. Finally, target node E is visited and searching algorithm is terminated after node expansion at D. The node expansion and paths are shown in Fig. 2.2.

In Dijkstra's algorithm, it takes long processing times if number of nodes are large because it visits the nodes in wave front in all directions. Therefore, A* algorithm

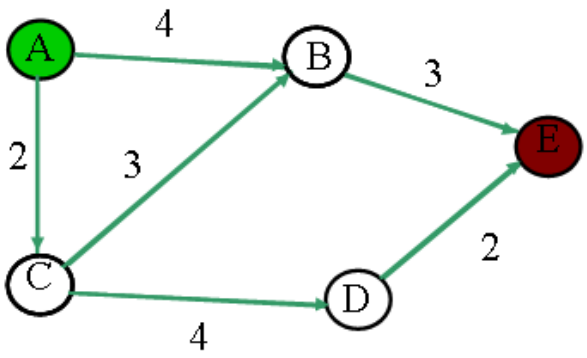


Figure 2.2: Paths finding in Dijkstra's Algorithm

has been proposed to overcome the Dijkstra's algorithm using heuristic function.

2.1.2 A* Algorithm

The A* algorithm combines features of uniform cost search and pure heuristic search to efficiently compute optimal solutions. A* algorithm is a best-first search algorithm in which the cost associated with a node is $f(n) = g(n) + h(n)$, where $g(n)$ is the cost of the path from the initial state to node n and $h(n)$ is the heuristic estimate or the cost or a path from node n to a goal. Thus, $f(n)$ estimates the lowest total cost of any solution path going through node n. At each point a node with lowest f value is chosen for expansion. The algorithm terminates when a goal is chosen for expansion. Due to compose with heuristic function $h(n)$, A* algorithm can greatly reduce path finding cost (decrease search area) than Dijkstra's algorithm. A* uses a best-first search and finds a least-cost path from a given initial node to target node. As A* traverses the graph, it follows a path of the lowest known heuristic cost, keeping a sorted priority queue of alternate paths along the way. A* search

only expand a node if it seems promising. It only focuses to reach the goal node from the current node, not to reach every other nodes. It is optimal, if the heuristic function is admissible.

A* works by maintaining two sets, one containing nodes which may be a next visited nodes in the path (called the 'open' set) and one containing nodes that have already been visited (called the 'closed' set). In A* algorithm; initially set a start node S.

(1) Put the neighbor nodes of start node S into 'open' set as unexpanded nodes

(2) If 'open' is not empty, the following steps are continued. Otherwise, search algorithm is terminated

(3) Remove a node n with minimum f value from 'open' set , and place it into a 'closed' set to be used for expanded nodes

(4) If n is a target node, exit successfully with updating path back along from n to S

(5) Expand node n, generating all its adjacent nodes with point back to previous node (node n)

(6) For every adjacent nodes n' of n;

a. Calculate $f(n')$

b. If n' is not include in 'open' set, then add it to 'open'. After that, assign the newly computed $f(n')$ to node n

c. If n' already exists in 'open' set, compare the newly computed $f(n')$ with that previously assigned to n'. If the new value is lower, substitute it for the old (n' now points back to n instead of to its predecessor), i.e., update the cost of getting to this node and to any successors that this node may already have. Then add node n' to 'closed' set.

(7) Go back to Step 2.

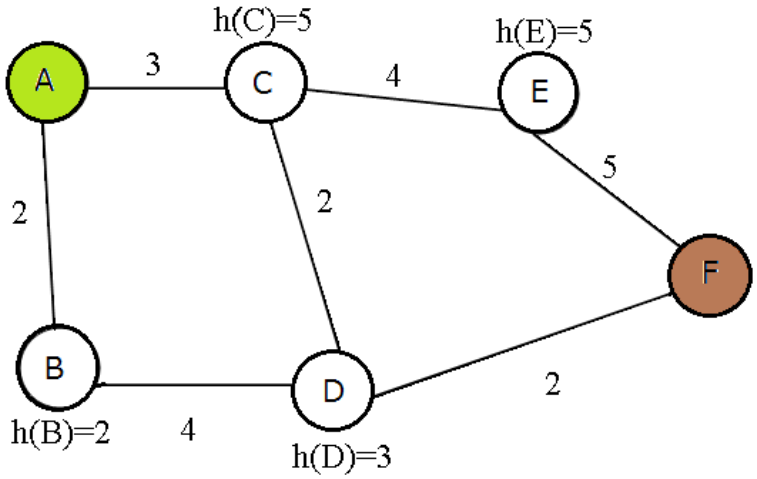


Figure 2.3: Example figure for A* Algorithm

Fig. 2.3 is an example of path finding in A* algorithm and it uses all algorithm steps described above. There are six nodes (A, B, C, D, E, F) in figure, A is start node and F is target node. The value attached between two nodes is the cost $g(n)$ of the path and the value attached each node is the estimated cost $h(n)$ of that node to target node. The cost function $f(n)$ is obtained by $g(n)$+$h(n)$, for example, if A is a start node, $f(B)$ of node B is $g(B)$+$h(B)$=4. Initially, set A as a start node and then all its neighbor nodes are added into 'open' set for node expansion orderly due to their cost function $f(n)$. In this figure, start node A has two neighbor nodes, node B and node C. The cost of node B ($f(B)$) is 4 and the cost of node C ($f(C)$) is 8. Therefore B is selected to expand and added to 'closed' set. In this time, B is not a target node. Therefore, neighbor nodes of B (node D) is added to 'open' list. However, C has lower function Cost (3+5=8) than D (6+3=9). For this reason, C

is selected to expand. The neighbor of C is E (7+5=12) and D (5+3=8) and D is selected for next step. However, D is exist in 'open' set already by node expansion at B. Therefore, an algorithm compares the old value of D via B (6+3=9) and new value of D via C (5+3=8). New value of D is lower than the old value, therefore the path and its function cost is updated at node D with a successor node C. After node expansion at D, finally reached to the target node F, a path is updated and a search is terminated. The paths which finding in an algorithm is shown in Fig. 2.4.

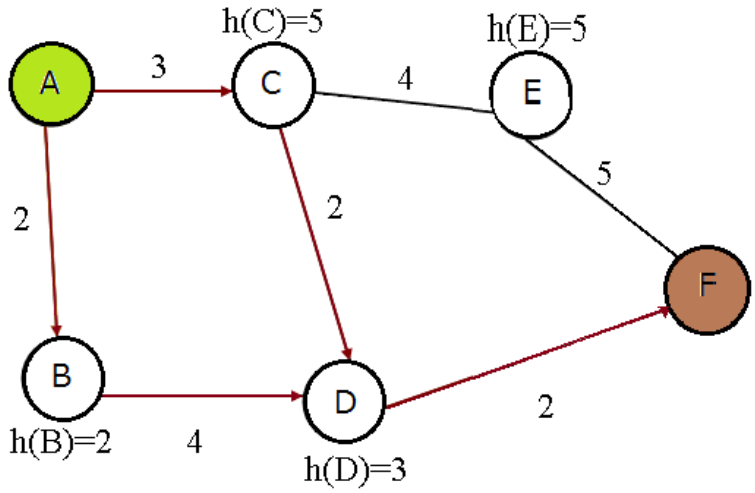
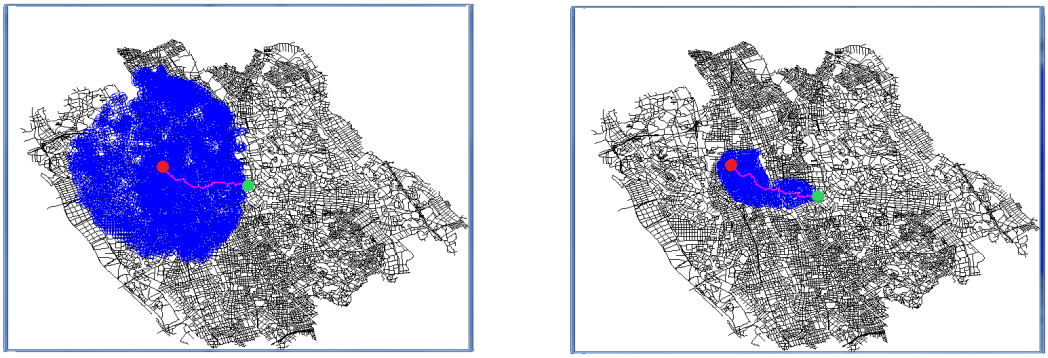


Figure 2.4: Obtained Paths in A* Algorithm

Fig. 2.5 compares the search areas of Dijkstra's algorithm and A* algorithm in real road map. In these figures, red dot represents the start point and green dot represents the destination point. And the expanding search area is shown in blue color. The red lines between the start and destination points show the shortest path finding by both algorithms. In Dijkstra's algorithm (see in Fig. 2.5(a)), the search

area is expanded in front wave for all nodes and that takes long processing time to find the shortest path. However, A* algorithm finds a path from a given start node to a destination node by employing a heuristic estimate. Each node is arranged by an estimate of the best route that goes through that node. A* visits the nodes only in order of the heuristic estimate and it decreases not only the search area but also the processing time (see in Fig. 2.5(b)).



(a) Dijkstra's Algorithm (b) A* Algorithm

Figure 2.5: Comparison of search area in two SPF algorithms

2.1.3 Materialized Path View

A shortest path query is a basic operation in several types of queries based on the road network distance, for example, kNN queries, ANN queries, R-kNN queries, and trip planning queries. Shortest path query algorithms have been studied since 1950's and several data structures and algorithms [12] have been proposed for this query. They can be categorized into, (1) methods compute-on-demand using adjacency list of nodes, and (2) methods used pre-computed optimal path.

Dijkstra's algorithm [8] and A* algorithm [13] are representative algorithms for the former type. A* algorithm is usually faster than Dijkstra's algorithm.

Materialized path view (MPV) approaches belong the latter type. It retrieves the shortest path by a lookup query in the pre-computed distance table. This method needs $O(n^2)$ space when the number of the nodes on the given graph is n. Therefore it is difficult to use when the network is large. Huang et al. [14] proposed semi-materialized method of the shortest path route to reduce the data amount. It only records the next pursued node along the shortest path, and the whole shortest path route is restored by tracking the next visiting node in sequence. Samet et al. [15] reduced the data amount to $O(n^{1.5})$, using semi-materialized approach.

The shortest path can be retrieved fast on MPV, however, it has a problem in a huge data amount as mentioned above. Therefore, several hierarchical representation methods have been proposed to reduce the amount of data. For example, Jing et al. [14] proposed the hierarchical encoded path view (HEPV) using hierarchical representation and semi-materialized approach. The principle of this method is partitioning a given graph G into several subgraphs SG_i. Distances between every two possible combination of nodes are calculated to compose a locally materialized distance table. Next, merging the neighboring subgraphs, it constructs the higher level subgraphs in a stepwise fashion. In a higher level, the distance table is built only for the border nodes between the subgraphs.

Hierarchical Materialized Path View

Efficient path computation is essential for applications such as intelligent transportation systems (ITS) and network routing [[16][17][18]]. In ITS navigation systems, many path requests can be submitted over the same, typically huge, transportation network within a small time window. While path precomputation (path view) would provide an efficient path query response, it raises three problems which must be addressed:

1) precomputed paths exceed the current computer main memory capacity for large networks;

2) disk-based solutions are too inefficient to meet the stringent requirements of these target applications;

3) path views become too costly to update for large graphs (resulting in out-of-date query results).

Hierarchical encoded path view (HEPV) model that addresses all three problems. By hierarchically encoding partial paths, HEPV reduces the view encoding time, updating time and storage requirements.

HEPV generates a hierarchical structure from a Flat Path View (FPV) structure based on fragmentation . Flat Path View (FPV) stores the all-pair shortest paths for a given graph. Because storing all shortest paths in their entirety requires an unrealistically large amount of storage, FPV only stores the origin, destination, direct successor node (called next-hop), and the weight for a shortest path for this O-D pair [[19][20]]. If there exist more than one shortest path between an O-D pair, each is stored for a different next-hop. The computation of the FPV, equal to calculating all-pair shortest paths, is computationally expensive, and the space requirement for the FPV is also high. For large maps, computing or updating the FPV may take a long time, therefore can only be performed with longer, possibly unacceptable, delays. Fig. 2.6 described the encoded flat path view structure.

HEPV generates a hierarchical graph from a flat graph based on fragmentation. It then pushes up all border nodes, the nodes that belong to more than one fragment, to generate a map at the next higher level. A higher-level map consists of only border nodes. Therefore it is a much more compact graph which represents all cross-partition points on the map at the level below. Fig. 2.7 described a simple example of creating a two level hierarchical graph from a flat graph.

After the hierarchical graph is created, HEPV generates a FPV for each frag-

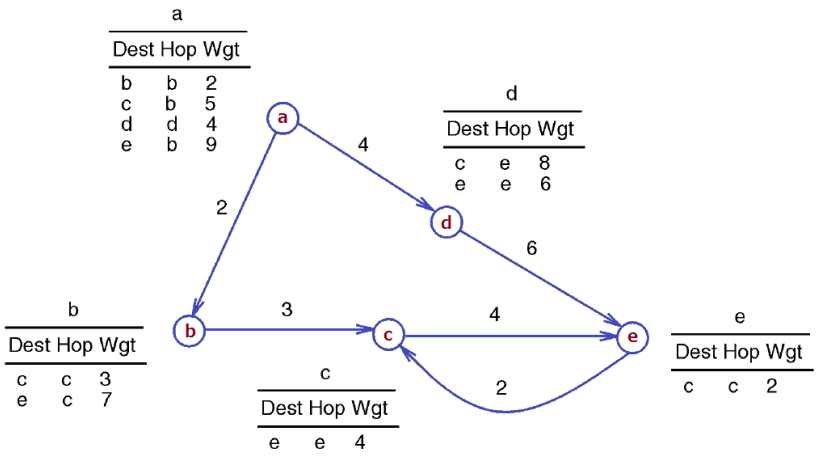


Figure 2.6: Encoded flat path view structure [14]

ment at all levels by precomputing all-pair shortest paths within this fragment. This collection of FPVs across all levels in the hierarchy is called the Hierarchical Path Views (HPV). And then the encoded HPV algorithm encodes the HPV for a hierarchical graph starting from the ground level upwards to the top-most level. At each level, it encodes all path views at this level by invoking an all-pair shortest path algorithm.

The hierarchical representation, such as HEPV, is suitable for the fast calculation of the shortest path between two points. However, the tables size in a higher hierarchy increase rapidly, then the total memory size of this structure becomes very large. Adding this, when a weight of the link is changed by a traffic accident or road maintenance, changing of weights (for example, distance) in the table affects a wide area in the table.

Jung et al. proposed another hierarchical materialized path view named HiTi graph [[21][22]]. This method also materializes distance between two nodes in the

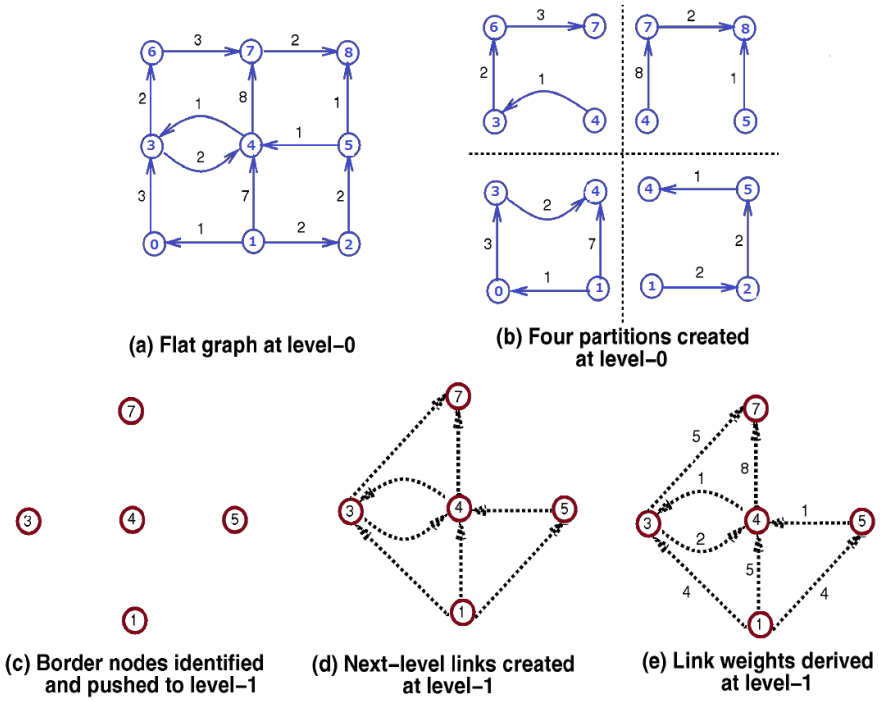


Figure 2.7: Two level hierarchical graph structure [14]

graph, and constructs the hierarchy. The big difference between HiTi and HEPV is that HiTi does not materialize in the leaf level subgraphs. Therefore, the total data amount of the HiTi graph is smaller than HEPV. The HiTi prunes the hierarchical tree leaves by using A* algorithm. Shekhar et al. [16] analyzed hierarchical-MPV in terms of the storage/computation-time trade offs. Their paper is closely related with our work, however, their investigation assumes the hierarchical structure essentially. The amount of data used in our methods decrease by comparing with the existing methods, HEPV and HiTi.

2.2 Real-time Monitoring for LBS

Location tracking of moving objects (MO) has been studied actively for such applications as wildlife animal monitoring, child-care systems, intelligent transport systems, logistics, and fleet management. These studies are roughly categorized into two groups: one targeting MOs that can move freely in the Euclidean space, and the other one targeting MOs for which the trajectories are restricted to a road network.

Although frequent reports improve the accuracy of monitoring MO position, it increases location update cost. To reduce this cost, several policies, including time-based [23], distance-based [24], dead-reckoning [25], and safe range adjustment [26] algorithms, have been proposed. The tracking method described in this paper belongs to the category of MOs restricted to a road network. Therefore, this will be the focus of the rest of this section.

The most of the efficient methods proposed in the literature are based on the prediction of movement of MO on a road network. Wolfson et al. [[27][28]] studied this problem under the assumption that objects move along a pre-specified route. They proposed location update policies based on several types of dead-reckoning. In their study, deviations from the pre-specified route are not considered. Tiesyte et al. [[29][30]] proposed a monitoring method targeting buses that strictly follow a predetermined route. However, their algorithm also does not allow any divergence from the specified route. Thus, these studies have different objectives than our study.

Ding and Güting [31] studied MO management on a road network. They assumed that objects could move freely on the road network. In their work, they proposed three location update policies: ID-triggered location update, distance-threshold-triggered location update, and speed-threshold-triggered location update. Čivilis et al. [[32][33]] proposed a tracking method for MOs based on a client-server architec-

ture. A client (MO) is equipped with a GPS receiver and a communication device. A client predicts its own position, and when the difference between the current and predicted positions exceeds a threshold value, the client sends an update to the server. In their work, three simple prediction policies were proposed. However, these studies do not use knowledge about objects' patterns of motion.

Spatio-temporal access methods to manage the moment-to-moment changes in an object's position have also been studied actively. Frentzos [34] proposed a spatio-temporal data structure for MOs on a road network. Ding et al. [31] proposed a suitable method for managing object positions on a road network. Usually, a road network is divided into several short segments that connect intersection to intersection. However, to manage the object positions on such short segments is not efficient. Thus, they proposed a data structure named Moving Objects on Dynamic Transportation Networks in which road segments are connected to form a route. The proposed FUR method follows this suggestion.

In addition to acquiring the MO positions, a spatio-temporal query method is also essential for real-time monitoring applications. Ku et al. [35] proposed a k-NN search for MOs. Mouratidis et al. [36] also proposed a k-NN MO search method. Hsueh et al. [37] proposed an MO management method using a location information table. These techniques are necessary to serve LBS based on MO monitoring.

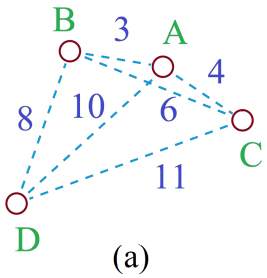
To create an MO monitoring method, a technique for matching a location acquired by a device such as GPS with a position on a road network is essential. This technique is called *map matching*, and has been actively studied in [[38][39][40][41]]. In this thesis, we applied the map-matching technique to acquire initial FUR, and to determine deviation from the FUR and the dead-reckoning route.

2.3 RkNN Queries in LBS

When a set of points P is given, a query to find the nearest neighbor of a point q ($\in P$) is called a nearest neighbor (NN) query. Many studies have been conducted

for nearest neighbor (NN) query based on Euclidean distances [[42][43][44]], network distances [[45][46][47][48]] and high dimensional distances [[49][50][51]]. Several efficient algorithms for NN query in spatial indexing structure [[52][53][54][55][56]] also have been proposed as the major importance of NN query in some areas, for example, databases and mining, statistics and data analysis, information retrieval, etc. Most well-know variants of NN queries are k-NN queries [[42][57][58][43][59][60]], k-NN join queries [61], approximate nearest neighbor (ANN) query [62], continuous nearest neighbor (CNN) queries [[36][63][64]], range NN queries [[65][66][67][68][69]] and reverse nearest neighbor (RNN) queries [[70][71]]. In this thesis, we propose an efficient algorithm for RkNN query on road network distances.

Fig. 2.8 is a simple example of nearest neighbors and reverse nearest neighbors. In Fig. 2.8, the values along the dotted line indicate the distances between two points. In this figure, the NN of A is B, and the NN of B is A. In this case, one NN is searched for; however, when a number k (an arbitrary number) of NNs are sought, the query is called a kNN query. Fig. 2.8(b) shows the 1NN and 2NN of each point in the figure.



(a)

	1NN	2NN
A	B	B,C
B	A	A,C
C	A	A,B
D	B	A,B

(b)

	R1NN	R2NN
A	{B,C}	{B,C,D}
B	{A,D}	{A,C,D}
C	∅	{A,B}
D	∅	∅

(c)

Figure 2.8: Example of an NN and RNN

Conversely, when q ($\in P$) is the NN of p ($\in P$), p is called a reverse nearest neighbor (RNN) of q. The result of an RNN query is given as a set. For example, if

A is the NN of B, then A is included in the RNN of B. To broaden the definition, when q is included in the kNN of p, q is called an RkNN of p. Fig. 2.8(c) shows the R1NN and R2NN of each point.

Query algorithms for RkNN have been actively studied on the basis of the Euclidean distance. The reverse nearest neighbor (RNN) query and its corresponding algorithm was first proposed by Korn et al. [70]. The RNN algorithm required precomputed data, in which the distance from each POI to its nearest neighbor (NN) is calculated. Next, POIs are registered in an R-tree with the distance to the NN; furthermore, the circle centered at a POI with the distance to the NN is called a vicinity circle. The RNN of the query point q is searched in the R-tree by searching the POIs in which vicinity circles overlap with q. However, this method is not suitable for RkNN query because the R-tree must be constructed with vicinity circles of predefined k-th NN distances, but the value of k is usually determined when a query is issued. Therefore, the distance to k-th NN cannot be determined when the structure is actually constructed.

Stanoi et al. [71] proposed an approach without precomputation called SAE. Tao et al. [72] proposed another efficient algorithm called TPL, which recursively prunes the search space using the bisector between query point q and its NN. These methods do not require any precomputation; therefore, they are applicable to general RkNN queries, however, these efficient methods cannot be applied to RkNN queries involving road network distances.

To obtain the results of several queries that compute road network distances without precomputation, Papadias et al. [1] proposed the following two methods: incremental network expansion (INE) and IER. They weighted the pros and cons of each method by applying them to range queries, k nearest neighbor queries, closest pair queries, and distance joins. INE searches POIs on the road network using Dijkstra 's algorithm [8] without any foreknowledge. In contrast, IER can apply a more efficient shortest path search algorithm, such as an A* algorithm [13]

and network distance materialization method [14], because two terminal points to determine the shortest path are given by the Euclidean distance search.

Yiu et al. [73] first proposed RkNN algorithms applicable to road networks. The basic idea here is that the area in which RkNN exists is searched by gradually enlarging the search area via Dijkstra's shortest path algorithm. They proposed two algorithms (called the Eager and Lazy algorithms) that differ in their respective pruning methods. Between these methods, the Eager algorithm searches RkNN significantly faster, especially when the value of k is small and POIs are densely distributed. In other words, it is efficient when the search area is small. In contrast, for large k or sparsely distributed POIs, it requires long processing times because the Eager algorithm gradually enlarges the search area, similar to Dijkstra's algorithm, and the kNNs are searched at every visited road network node. To cope with this performance problem, Yiu et al. also proposed a path materialization method. In addition to the work of Yiu et al. Safer et al. [74] proposed algorithms using network Voronoi diagrams. Cheema et al. [75] proposed an RkNN algorithm on a road network.

In location based services (LBS), reverse nearest neighbor (RkNN) query is required in a wide variety of applications, including facility management, taxi allocation, decision making and marketing plan, etc. Therefore, we build upon the RkNN algorithm and materialized path views (MPVs) which is suitable for LBS. The next section is described about the spatial indexing method which is one of the important function in spatial queries.

2.4 Spatial Indexing Method

2.4.1 General indexing methods used in spatio-temporal databases

As it is obvious, the improvement of a spatio-temporal access method suitable for the moving objects on the road networks is a very attractive challenge be-

cause a great number of real-world spatiotemporal database applications have to work mainly with objects (e.g. cars) moving on the road networks. In both typical queries and complicated queries, for some examples, range queries [[76][77]], nearest neighbor queries [[78][79][80][81][82][83]] as in typical queries and such kind of complicated queries as skyline queries [[84][85][86]], k nearest neighbor queries [[87][88][89]], Reverse k nearest neighbor queries [[70][71][72][73] etc., needed indexing method to manage spatio-temporal information. Much work has been recently conducted in the domain of indexing spatiotemporal data and several spatiotemporal access methods have been proposed, B-tree[90], the B^+ tree[91], R-tree structure and variants [[92][93][94][95]], 3D R-Tree [96], HR-Tree [97], TB-Tree, STR-Tree[98] and so on. Among all indexing methods, R-tree is basically used in spatial query processing.

2.4.2 R-tree

R-tree is most popular tree data structure used for spatial access methods, for example, indexing multi-dimensional information such as geographical coordinates, rectangles or polygons and so on. An R-tree is a height-balanced tree similar to a B^+ tree[91] with index records in its leaf nodes containing pointers to data objects. A complex spatial object is represented by minimum bounding rectangles while preserving essential geometric properties. The index is completely dynamic, i.e, inserts and deletes can be intermixed with searches and no periodic reorganization is required. The basic idea of the data structure is to group nearby objects and represent them with their minimum bounding rectangle (MBR); the 'R' in R-tree is for rectangle. As mention earlier, R-tree is height-balanced search tree, therefore all leaf nodes are at the same level in R-tree. All entries in R-tree has an index record of the form

$$I, tuple - identifier$$

where $tuple-identifier$ refers to a tuple in the database and I is an n-dimensional rectangle which is the bounding box of the spatial object indexed

$$I = (I_0, I_1, ..., I_{n-1}).$$

Non-leaf nodes in R-tree have a structure of entries as;

$$I, child-pointer$$

in which $child-pointer$ is the address of a lower node in R-tree and I covers all rectangles in the lower node's entries.

If M is the maximum number of entries fixed in one node, $m <= M/2$ should be the minimum number of entries in a node. Properties of R-tree are;

(1) Every leaf node contains between m and M index records unless it is the root

(2) For each index record $(I, tuple-identifier)$ in a leaf node, I is the smallest rectangle that spatially contains the n-dimensional data object represented by the indicated tuple

(3) Every non-leaf node has between m and M children unless it is the root

(4) For each entry $(I, child-pointer)$ in a non-leaf node, I is the smallest rectangle that contains all rectangles in the child node

(5) The root node has at least two children unless it is a leaf

(6) All leaves appear on the same level.

Fig. 2.9 and Fig. 2.10 show the tree structure and minimum bounding rectangles (MBRs) in R-tree. The search in R-tree descends the tree from the root. Every internal node (non leaf node) contains a set of rectangles and pointers to the corresponding child node and every leaf node contains the rectangles of spatial

entries.

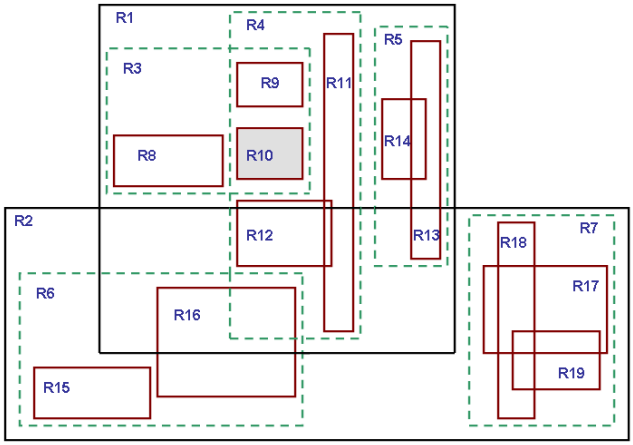


Figure 2.9: Searching in MBRs

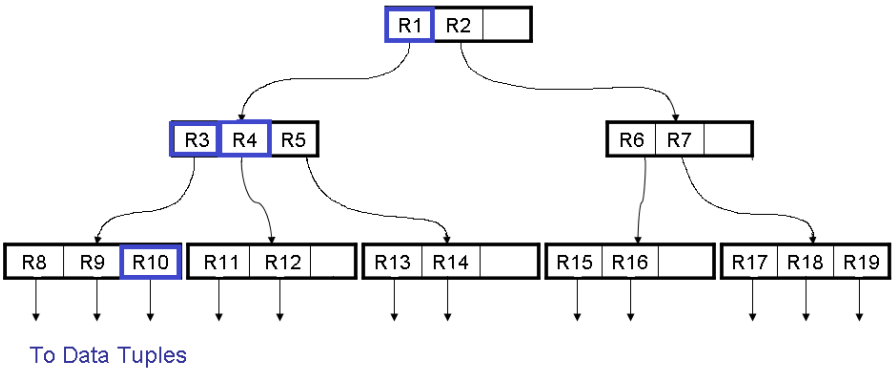


Figure 2.10: Searching in R-tree Structure

(1)[Search subtree] If T is not a leaf, check each entry E to determine whether EI overlaps S. For all overlapping entries, invoke search on the tree whose root node is pointed to by E_P.

(2)[Search leaf node]If T is a leaf, check all entries E to determine whether EI overlaps S. If it is correct, E becomes a result record.

In Fig. 2.9, assume that search entries are in rectangle R10, then the desire rectangle R10 is overlapped in root entry R1. For this reason, the algorithm checks all entries of R1 and finds overlapped rectangles with a search rectangle R10. In this step, R3 and R4 is overlapped by search rectangle R10. T is not still leaf node, therefore search in subtree R3 and R4 is continued. Finally, the search reaches to the search rectangle R10 as the result. Fig. 2.10 shows the searching in tree structure and the rectangles in blue color indicates the searching order in overlapped MBRs. Now T is a leaf node, therefore check all entries where it is cover with result rectangle or not. And if it is correct, record the result index entries and search is terminated. Due to its effectiveness, for example, query indexing applied in IER adaption, R-tree becomes most useful indexing method for spatial queries in LBS.

CHAPTER 3

Real-time Monitoring of Moving Objects Using Frequently Used Routes

3.1 Real-time Monitoring of Moving Objects

Recently, real-time monitoring of moving objects (MOs), such as cars and humans, has attracted interest. In these systems, tracking trajectories and positions of MOs accurately and for the least communication cost is an important goal. To achieve this, MO position prediction is important. Some studies have proposed real-time monitoring systems that share route information between MOs and a server to predict MO position; however, these systems target public transportation, such as buses, for which the route is always fixed. To the best of the our knowledge, a real-time monitoring method for sharing route information between passenger cars and a central server does not exist. This paper proposes such a method using the sharing of frequently used routes (FUR) data between each MO and the server. The server and the MO predict near-future position of MO on the basis of the FUR information using a common algorithm. When the position of the MO deviates from the prediction, it sends a message to the server to maintain position synchronization.

In this thesis, we evaluate the advanced method using real trajectories of cars and shows that the proposed method outperforms the conventional method, that is, dead-reckoning on a road network.

3.2 Preliminaries

Efficient transportation systems for human travel and logistics are required for both environmental and resource-saving reasons. In addition, the diversification of information services required for comfortable and efficient car driving has increased. In particular, the popularity of in-car navigation systems is increasing, as well as pedestrian navigation systems using mobile phones. Using these devices, it is possible to track moving objects and deliver to them various types of location-based services (LBS).

Car position monitoring is an essential technology for management of taxis, home delivery vehicles, patrol cars, and ambulances . Such moving objects (MOs) can get their positions easily by using GPS (Global Positioning System). If MOs send their position to a server every T_u seconds, the server can monitor the positions of the fleet in real time. However, this method requires very frequent reports to maintain high accuracy of the MO positions. On the other hand, when T_u is large and an MO moves a large distance during T_u seconds, the server cannot know the real trajectory of that MO during this interval. However, if the trajectory information is known, then the state of traffic congestion on the road where the MO is moving can be estimated. Therefore, efficient real-time monitoring of MO position and trajectory is required .

To reduce the overhead of position updating, several shared-prediction-based approaches have been proposed [99] . These approaches share a prediction of MO's near-future position between each MO and the server. The MO sends a message to the server when the MO's real position deviates from the predicted position. When the server receives the message, it modifies the prediction based on the parameters

in the message and then continues monitoring.

Over the previous two decades, several methods have been reported for the prediction of MO movement; however, most of these have targeted MOs freely moving in Euclidian space. The most common applications are tracking for cellular phone networks and LBSs, which are not constrained to movement on a road network.

Scheduled vehicles, for example buses, are an important application target of real-time monitoring. In this case, the MO moves strictly along the predetermined route, and therefore the arrival time to bus stops can be easily estimated from the schedule and deviations from the route do not need to be considered. Late and early deviations from the schedule are the only cases that require a location update to be sent to the server.

Dead-reckoning on a road network is the second method for MO position prediction. This method predicts MO near-future position based on the assumption that the MO is moving along a road with a constant speed. When the MO reaches the end of the road or a T-junction, the MO sends the information about the newly selected road.

The third method uses prediction from the trajectory history of the individual MO. For example, a delivery car that goes around to franchise stores can be assumed to follow a similar route every day. By accumulating the routes day by day, the car's position can be predicted using this frequently used routes (FUR) information. The proposed method is based on this observation. This idea is straightforward, and Liu et al. [100] have proposed such a trajectory prediction method based on FUR information. However, their work is limited to future trajectory prediction to serve LBS, and they do not apply this method to prediction for the real-time monitoring of MOs.

The contributions are the following:

- This is the first work to adopt FUR information to real-time monitoring of

MO locations.

- The adaptive Level of Detail (LoD) setting for monitoring has been proposed.
- Its efficiency was demonstrated experimentally and we show that the proposed method outperforms dead-reckoning on a road network by the results from experiments using real movement paths.

3.3 Moving Object Monitoring with FUR

3.3.1 System configuration

This section describes the fundamental ideas of the proposed real-time monitoring system. Fig. 3.1 shows the configuration of the system. A MO has a terminal equipped with a GPS receiver, a wireless communication device, a road map, and a computer. This configuration consists of almost the same components as the in-car navigation systems common in Japan, except for the wireless communication device. Due to the requirement of road map data inside, large storage capacity is required for MO, therefore, we defined it as a thick client.

The MO acquires its position every 1 second using the GPS, and this position is matched to the map to determine the position on the road network. The MO also predicts its own position. The predicted position is compared with the position matched on the map, and the difference between them is calculated. If the difference exceeds a predetermined threshold value (θ_D), the MO sends a message to the server. If the difference is less than θ_D, the MO does not send any message to the server and continues prediction and map matching.

The server monitors several MOs simultaneously. The monitoring of an MO starts when the server receives the start-of-trip message from the MO, after which the server starts a thread for monitoring the MO. In the thread, the location of the MO is predicted by the same algorithm and the parameters of the MO. Therefore,

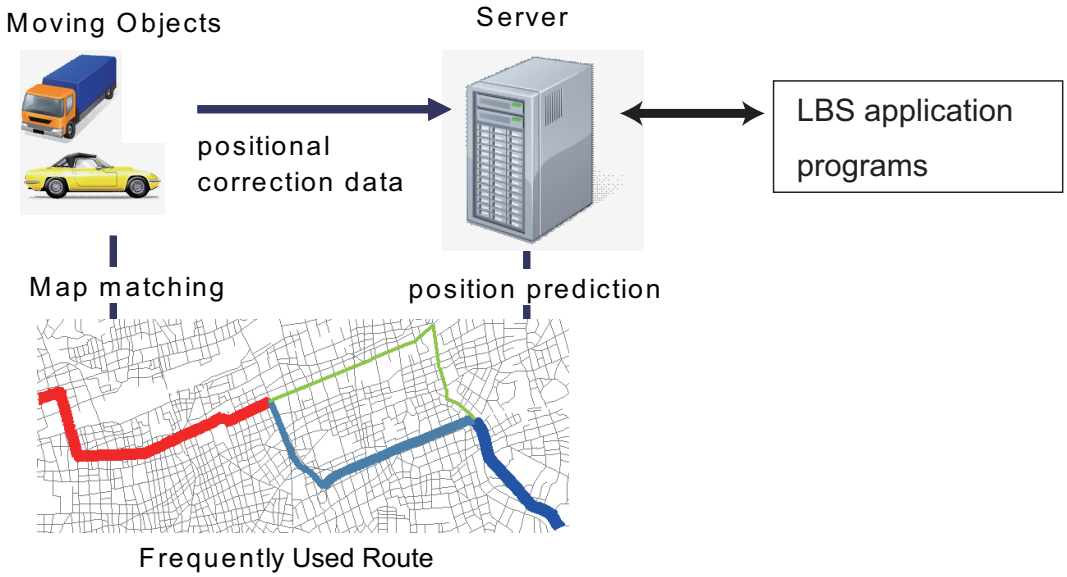


Figure 3.1: General concept of the proposed system

the prediction is synchronized between the MO and the server. When the server receives a message (usually a position correction) from an MO, the server corrects the parameters for the MO, and then continues the prediction.

In parallel with this monitoring, the server also sends MO locations to the LBS application programs. The sent locations have uncertainties which depend on the threshold θ_D. When a small θ_D is specified, the communication cost is likely to be high. Usually, a large θ_D is specified, for example 1 km. When the application program requests a more precise location, the server sends a message to MOs to decrease θ_D. This operation is called the "LoD (Level of Detail) control". LoD control is necessary to control some location requests that need more precision and also occasionally when communication capacity becomes limited. Usually, high LoD is requested only for a limited number of MOs, for example, for MOs that are inside a special area of the town or neighborhood of a designated position.

For location prediction on a road network, the road segments along which each MO is passing must be determined. This is determined by two methods, one based

on FUR and the other based on dead-reckoning on the road network.

3.3.2 Frequently used routes

When one drives a car daily, a trip starts from one location, then ends at another location. For example, one can start the trip from home and go to a shopping center, stay there for some time, and then go to the office. During this trip, one may stop at a gas station for refueling or stop at a convenience store to buy a magazine. Hereafter, we call the start location of a trip or any place at which the MO stays for long periods of time a "Base Point (BP)". In the above example, home, the shopping center, and the office are BPs. Furthermore, places at which one stops for relatively short periods of time are called "Points of Interest (POIs)". In the same example, the gas station and the convenience store are POIs. To summarize, a trip starts at a BP aiming for another BP, with some POIs being visited during the trip.

It can be assumed that there are several target BPs for a trip starting from an origin BP. For example, when a trip starts from home, it can be assumed that the office, the usual shopping centers, and favorite restaurants may be target BPs. In daily life, the number of possible target BPs can be assumed to be limited. In a daily drive, the drivers are likely to take their favorite routes between two BPs, and so the routes can be assumed to be similar. The proposed method is based on this assumption.

As another example, a delivery car going around franchise chain stores would consider the office, the warehouse, and the garage as BPs, and the stores as POIs. On a given day, it can be assumed that the driver takes a similar route connecting the stores in order because the driver knows the most efficient or comfortable roads for driving; however, deviations due to traffic jams, road repair, or a sense of adventure may occur. Hereafter, we call a set of historical routes starting from a BP a FUR.

Fig. 3.2 shows an example of a FUR. The FUR consists of a series of BP to BP paths. Each FUR has a start BP (S in this example) and several destination BPs (B, C, and D). The thickness of roads shows the frequency at which the given route is used. Thus, this figure shows that the car often goes to B from A and only occasionally to C or D.

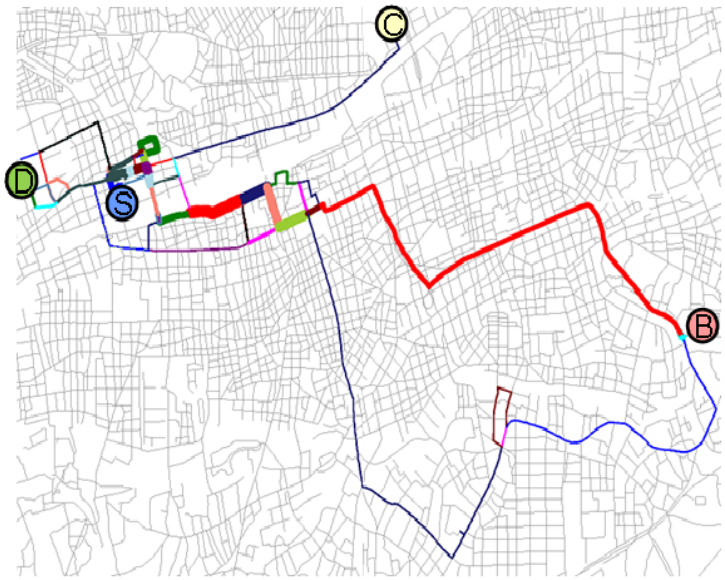


Figure 3.2: Example of frequently used routes

3.3.3 FUR description

A FUR is initially acquired using one of the methods described below. When many historical trajectories of an MO are available, the FUR can be extracted from this information. When an MO has a recommended route, for example a shortest route or a route that is easy to drive, this can be set as the initial FUR. The most probable situation is incremental acquisition starting from an empty FUR.

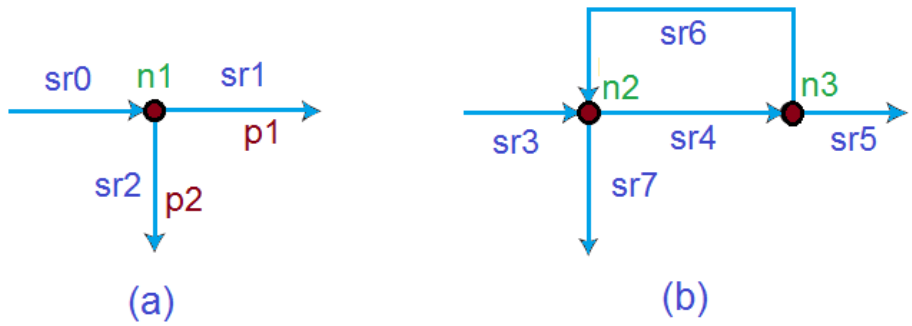


Figure 3.3: Construction of a FUR

Independent of how the routes in the FUR are acquired, an MO can have many different FURs, each with a different starting BP. Once the starting BP is specified, the FUR is determined. Usually, a FUR is not only a tree rooted at the starting BP, but also a network that contains closed loops. This network is a subgraph of the background road network, which consists of a graph whose nodes represent intersections and links represent road segments.

Because the length of each road segment connecting intersections is not long, a route can contain an enormous number of road segments. Therefore, a simple expression is used to describe the formation of the route network. A FUR has branching paths and merging paths. The intersections at which these paths branch or merge are called *control nodes*. The network denoting a FUR consists of BPs and control nodes. A path on the network connecting the control points with each other or between a control point and a BP is called a *subroute*. A subroute consists of several road segments.

Fig. 3.3(a) shows an example of the branches in a FUR network. In this figure, $sr0$, $sr1$, and $sr2$ indicate the subroute links, and $p1$ and $p2$ indicate the probabilities that the subroute will be taken if the MO arrives at node $n1$ from $sr0$. At node $n1$, subroute $sr0$ is followed by sr1 with probability $p1$, and turns right to $sr2$ with probability $p2$. Fig. 3.3(b) shows a more complicated example. At node $n2$, the subroute $sr3$ branches into $sr4$ and $sr7$, and the subroute $sr4$ branches again at node $n3$ into $sr5$ and $sr6$. The subroute reaching node $n2$ from $sr6$ always continues to $sr7$. As this example shows, a subroute should be expressed by a directed graph that includes the probabilities of the next selected subroute for a given trip. Karimi and Liu [101] proposed a method using probability matrices to express the trajectory choice at each intersection. They considered all combinations of the coming links and going links at each intersection. On the other hand, the proposed method can reduce the number of combinations and still obtain an expression of the trajectory choice. On the FUR network, the intersections that need an expression of the trajectory choice are restricted to the FUR network nodes, that is, the control nodes.

Table 3.1: Subroute structure

SR_ID	subroute ID
v	average velocity
SN	start node ID of subroute
EN	end node ID of subroute
f	choosing frequency of SR_ID
n	number of links
Vtx[0]	vertex ID[0]
...	...
Vtx[n–1]	vertex ID[n–1]
Dist[0]	distance from SN to Vtx[0]
...	...
Dist[n–1]	distance from SN to Vtx[n–1]

The subroutes in the network are expressed in the format shown in Table 3.1. In

this table, SR_ID is the ID assigned to the subroute. SN and EN are the start node ID and the end node ID of the subroute, and f is the frequency with which the subroute is selected. Each subroute is assigned the average velocity (v) of the MO. Vtx[i] is a vertex ID forming the subroute. Dist[i] expresses the distance along the road in meters from the start of the subroute (SN) to Vtx[i]. This value is used for rapid calculation of the distance between the MO's current and expected positions on a FUR.

Environments

For the experiments, a digitized road map from a 1/25,000 scale base map that covers Saitama City, Japan was used. The trajectory of the MO was captured by a GPS logger (Global Sat, BT-335) placed in the car. The GPS logger recorded longitude, latitude, and time of measurement every 1 second.

Many trajectories were collected using the GPS logger for 1 year. These trajectories are mainly the commuting route from the home of one of the authors to the office (about 10 km). Most of the trips were made from home to the office between 08:00 and 09:30 and from the office back home between 21:00 and 22:30. Fig. 3.2 shows an example of the FUR obtained from the accumulated log by map matching.

3.4 Moving Object Tracking Algorithms for thick client

3.4.1 Moving object tracking for thick client

As mentioned earlier, the MO and the server share FURs and the algorithm predicting the MO's position. Each MO continuously acquires its current position using GPS measurements. Thus, an attempt is made to match the position of an MO with the current predicted position on the route. When late arrival, early arrival, or deviation from the predicted route is detected, the MO reports it to the server. When the server receives this message from the MO, the server can capture the

position of the MO within a predetermined allowance.

Two prediction modes are used in the algorithm. One predicts the position based on a FUR (*On Route (OR) mode*), and the other one is dead-reckoning on the road network (*Dead-Reckoning (DR) mode*). Tracking of MOs primarily begins in the OR mode. Then, when an MO diverges from the FUR, the prediction mode is shifted to the DR mode. The outline of the prediction algorithm is as follows:

(1) Match the object's position with the most frequently used subroute. Find the distance between the current and predicted positions on the route. If this exceeds the threshold value due to late or early arrival, the MO sends its current position to the server for position synchronization.

(2) When the MO diverges to another subroute on a FUR of lower trip frequency, the MO reports to the server the newly selected subroute for synchronization.

(3) When the MO diverges to a road segment that is not on a FUR, both the server and the MO predict the route and the position using dead-reckoning on the road network.

Usually, dead-reckoning predicts the future position under the assumption that the MO continues in the same direction at the same speed. However, under the conditions dealt with in this paper, the locus of MOs is restricted to the road network. Thus, dead-reckoning is restricted to the road network (dead-reckoning on the road network: DRRN). Namely, the MO moves in the same direction along the road until it reaches a dead-end or a T-junction. There, the MO sends information about its change in direction to the server, and then continues with dead-reckoning. Ding and Güting [31] and Čivilis et al. [32] also use a similar prediction method. The following summarizes the method used for dead reckoning.

- An MO's motion is restricted to the road network.

- When an MO encounters an intersection, it selects the road segment going straight ahead.
- When an MO encounters a T-junction, DRRN is suspended until the road link on which the selection of MO can be determined.

It is often the case that a road does not have a unique name. This is especially common in Japan, but rarer in most European and North American countries. Thus, DRRN is defined as above. If each road has a unique name, DRRN can be defined as an MO continuing on the same named road at a constant speed.

As described above, the server and the MO share the FUR, the prediction algorithm, and the parameters. When the distance between the MO's actual and the predicted positions exceeds a threshold value or the MO deviates from the predicted route, the MO sends the current parameters and maintains the synchronization with the prediction.

3.4.2 Moving object side algorithm for thick client

Algorithm 4 shows the procedure executed on the MO side for following the above-described method. In this algorithm, the *send* function sends a message to the server. The first parameter of *send* is op-code, the meaning of which is shown in Table 3.2.

Each car (MO) is assigned an individual ID to identify the car uniquely in the system. *getPosition* in the first line returns the location (p) of the MO captured by GPS, and *getBPID* searches for the BP that is the nearest neighbor of p and returns its ID. *getFUR* in the third line reads the FUR whose starting BP matches $BP\#$. Then, the most frequent subroute (rt) starting from $BP\#$ is determined from the FUR. Then, the MO sends op-code 0 with the current position to the server to signal the beginning of the trip (Line 4). *mode* in line 5 shows the tracking mode that takes either the value OR (on route) or DR (dead-reckoning). At the beginning

Table 3.2: op-codes

code	meaning	parameters
0	start of trip	carID and BPID
1	send position (OR-mode)	carID and position
2	another subroute is selected (OR-mode)	carID, position, and subrouteID
3	enter to dead-reckoning mode	carID,new RoadID, direction, and position
4	send position (DR-mode)	carID and position
5	another road segment is selected (DR-mode)	carID, new RoadID, direction, and position
6	recover to OR-mode	carID, position, and subrouteID
7	end of trip	carID and position

of the tracking, the mode is set to OR. As described later, when the server receives this message, the server also starts the monitoring of the MO. Lines 6 to 41 are repeated until the end of the trip.

The position of the MO is updated every 1 second (Line 7). The position is checked to determine whether it is still on the predetermined subroute. $OnRoute(p, rt)$ in line 9 does this check and returns a Boolean value, where the value **true** shows the MO's current position p is located on rt, and **false** shows p has deviated from the subroute rt. When the result is **true**, the distance between p and the predicted position of rt at the current time is calculated by the function $distNow(p, rt)$. Then, if the distance d is greater than a predetermined threshold value (θ_D), the MO informs the server that it is late or early with respect to the expected time of arrival.

Lines 15 to 21 correspond to the situation of the MO deviating from the predicted subroute, usually at an intersection. When p diverges from the currently predicted route, a new predicted route is determined by the function $getSubroute(p)$ (line 15). $getSubroute(p)$ first checks whether p is adjacent to the end node (EN) of the subroute. If it is, other subroutes connected to EN are retrieved, and the subroute

(sr) to which p can be best matched p is determined. When a matching subroute (or any subroute connected at the intersection) does not exist, *getSubroute* returns NULL.

If the search for a branch to which p can be matched on a FUR succeeds, the MO informs the server about the route change (line 17), and the MO's position is matched to this newly selected subroute(rt). On the other hand, when rt is NULL (i.e., the search fails), the tracking mode shifts to DR mode. *getDRRoute* returns a tuple consisting of rt, *newRoad#*, and dir, where rt is the path for dead-reckoning composed of the chain of road segments which has the best match with the position p, *newRoad#* is the uniquely assigned road segment ID on which dead-reckoning is started, and dir is the direction of movement on the road segment. To inform the server of this mode change, op-code 3 with car#, newRoad#, and position p is sent (line 21).

Lines stating from line 25 of the algorithm correspond to DR mode. In this mode, the MO's position is predicted based on the assumption that the MO continues straight ahead on the road. As with OR mode, whether the MO is still on the predicted route is checked (line 25). When p is on the route (rt), the difference between the MO's current position and the predicted position is calculated. When the difference exceeds the specified value θ_D, the MO sends the real position to the server (line 28). When the server receives this information, it corrects the MO's current position. When the MO diverges from the current route inferred by dead-reckoning, the part of the route from the start of rt to the current intersection is added to the FUR (line 31). This function also returns a Boolean value. **true** indicates that the MO has returned to a known FUR, in which case, the mode is switched to OR mode. On the other hand, when the return value is **false**, DR mode continues.

Algorithm 1 Moving Objects

```
1: p ← getPosition()
2: BP# ← getBPID(p)
3: rt ← getFUR(BP#)
4: send(0,car#, BP#)
5: mode ← OR {On Route}
6: repeat
7:    p ← getPosition()
8:    if mode = OR then
9:       if OnRoute(p,rt) then
10:         d ← distNow(p, rt)
11:         if d > θ_D then
12:            send(1,car#,p)
13:         end if
14:      else
15:         {rt, rt#} ← getSubRoute(p)
16:         if rt is not NULL then
17:            send(2,car#,rt#) {rt# is the selected subrouteID}
18:         else
19:            mode ← DR {Dead-Reckoning}
20:            {rt, newRoad#, dir} ← getDRRoute(p)
21:            send(3,car#,newRoad#, dir, p)
22:         end if
23:      end if
24:   else
25:      if OnRoute(p, rt) then
26:         d ← distNow(p, rt)
27:         if d > θ_D then
28:            send(4,car#,p)
29:         end if
30:      else
31:         if addNewLinkToFUR(rt) then
32:            {rt, rt#} ← getSubRoute(p)
33:            mode ← OR
34:            send(6,p,rt#)
35:         else
36:            {rt, newRoad#, dir} ← getDRRoute(p)
37:            send(5,car#,newRoad#, dir, p)
38:         end if
39:      end if
40:   end if
41: until end of trip
42: send(7,car#)
```

Then, the newly rt is searched (line 36). Next, the MO sends the newly selected road ID, the direction of motion(dir), and the current position to the server (line 37). When the MO reaches a T-junction or cannot find a road segment going straight, the MO also sends this format of signal to the server.

When the trip ends, the MO sends OP-code 7, which terminates the monitoring (line 42).

For simplifying the description of the algorithm, the frequency update for selected subroute is omitted. However, every time a new subroute is selected on the FUR, the frequency attached to the subroute is incremented.

3.4.3 Server side algorithm for thick client

Monitoring the system on the server side consists of three kinds of modules. One is the *communication module*, which monitors the data sent from each MO. When the server receives an op-code 0 message, this module starts a thread that tracks a new MO, which is a *tracking module*. The communication module distributes the messages from each MO to the corresponding tracking module. The third module is a *location observer* to provide the MOs' position to several types of LBS applications. This module receives the individual MO route and the latest positions, and responds to LBS applications based on their requests, for example, range query or k-NN query. This module also directs the communication module to alter LoD according to the LBS application's requests.

Algorithm 3 shows the pseudocode carried out on the tracking module. This module is started with two parameters, car# and BP#. This module retrieves the database, gets FUR for car# starting from BP#, sets the current car position at the BP# position, and selects the initial route that has the highest frequency on the FUR. This tracking module watches the data received from the MO and alters the MO's state according to the received data.

Algorithm 2 Tracking Module

Require: car# and BP#

1: Retrieve the FUR of the car# starting from BP#, then set the result to tr.
2: $cp \leftarrow$BP#'s position
3: $mode \leftarrow$ OR {On Route}
4: **repeat**
5: $\{code, args\} \leftarrow$receive()
6: **if** $code = 1$ **then**
7: $cp \leftarrow$reported position by $args$.
8: **else if** $code = 2$ **then**
9: Set tr to the new subroute specified by the SubrouteID in $args$.
10: $cp \leftarrow$reported position by $args$.
11: **else if** $code = 3$ **then**
12: $mode \leftarrow$ DR {Dead-Reckoning}
13: $tr \leftarrow getDRRoute(args)$
14: $cp \leftarrow$reported position by $args$.
15: **else if** $code = 4$ **then**
16: $cp \leftarrow$reported position by $args$.
17: **else if** $code = 5$ **then**
18: $addNewLinkToFUR(rt)$
19: $tr \leftarrow getDRRoute(args)$
20: $cp \leftarrow$reported position by $args$.
21: **else if** $code = 6$ **then**
22: $mode \leftarrow OR$
23: $cp \leftarrow$reported position by $args$.
24: Set tr to the new subroute specified by SubrouteID in $args$.
25: **end if**
26: send car#, tr, $currentTime$, cp to the location observer.
27: **until** code = 7

The individual tracking module in charge of each MO sends a tuple that consists of tr, ct, and cp to the location observer every time when the tracking module receives a message from the MO. tr is the route along which the MO will progress, ct is the current time, and cp is the current position when the update is received. tr and cp are altered based on the messages from the MO.

The location observer integrates the message sent from the tracking modules and provides the MO locations to LBS applications, depending on their request. This

module estimates the current position of MOs periodically (e.g., every 1 minute) based on tr, ct, and cp of each MO.

3.4.4 Experimental results of thick client model

Position monitoring

The communication cost of the proposed method was compared with that using dead-reckoning on road network (DRRN) for the entire trip to evaluate the efficiency of the proposed method. In the method described in Section 3.4.1, MOs send a small amount of data for all eight types of information. It was assumed that one communication requires only one packet, and therefore, the efficiency was evaluated using the number of packets sent.

First, the FUR information described in Section 3.3.3 with about 100 trajectories was obtained. Next, the MO position monitoring using the other 70 trajectories that were not used to create the FURs were tested. Figure **??** compares the proposed method and DRRN. In the proposed method, the speed obtained from the historical data can be used in OR mode. However, DRRN lacks this information. Thus, the MO's speed was varied from 10 to 30 km/h in increments of 10 km/h.

The horizontal axis in Fig. 3.4 is the permissible positional error (θ_D), and the vertical axis is the average number of communication packets needed to keep tracking under a θ_D value from 100 m to 1 km. As shown in this figure, though the best result was obtained when the speed was set 10 or 20 km/h, the packet number with dead-reckoning is not so sensitive to the MO's speed, especially when the θ_D value is large. This is because most packets in DRRN are only used to report road changes. Our proposed method outperforms the conventional method DRRN by factors from 2 to 4. As described in subsection 3.3.1, the positional update threshold value θ_D usually can be set with a large value, and then the precise location is requested for a limited number of MOs to satisfy the needs of LBS. This

control can be performed by a suitable setting of LoD. Whenever this assumption is true, the update cost can be quite low.

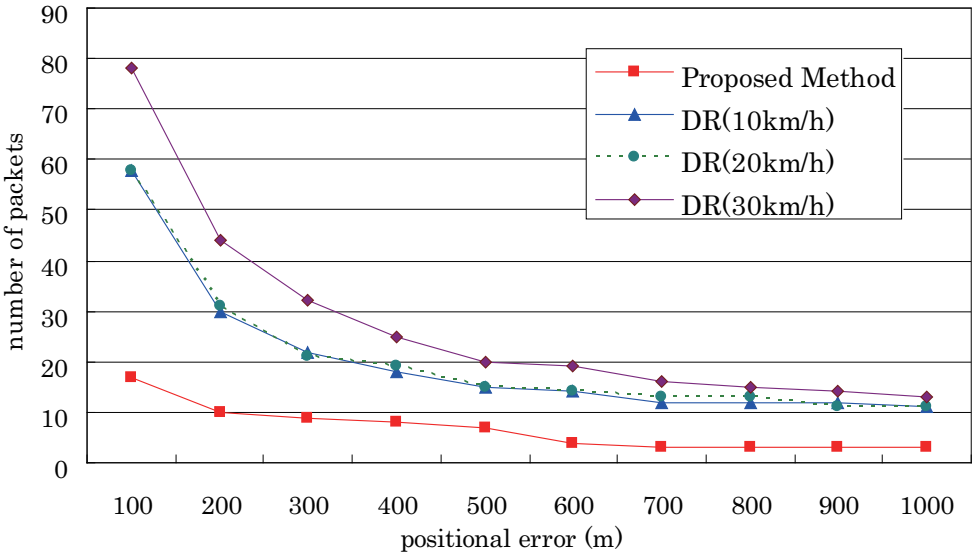


Figure 3.4: Comparison of communication cost

In this proposed method, it was assumed that the MOs possess a road map and sufficient computation power. However, the equipment that required in this system is expensive. Therefore, research into developing appropriate economical terminals, for example, equipped only with a GPS receiver and a communication device but without the road map data inside, is needed.

3.5 Monitoring Algorithms for thin client

The popularity of car navigation system as well as pedestrian navigation system using mobile phones has led to an interest in studying of efficient real time monitoring. In this session, we describe about a real time monitoring method aiming for thin client (e.g. mobile phones)which has small data storage size and convenience to use.

In the rich client model, both client and server sides share the route information which MO often used and predict the MO position by the same algorithm. The difference between the actual position obtained by the GPS receiver and the predicted position exceeds a predefined permissible error, or when it deviates from a predicted route, MO sends a message to the server. At the server side, based on the information sent from a MO, the parameter of MO position is corrected and it has been answered to the various inquires. The equipment that required in rich client model is expensive and it is impossible to have same road map in both server and client sides. For these reasons, we further develop the rich client model method in thin client model which MO has a terminal like a mobile phone with GPS receiver. The main differences of thin client and the previous rich client are the existence of the road map, storage size and the less of prediction route of MO in thin client side.

In thin client model, it does not exit road map data inside, so it is necessary to receive the route information from the server. For this reason, comparing with a rich client model, the communication times are increased in thin client model. On the other hand, a thin client model should just be a terminal like a mobile phone and it can perform real time monitoring in very simple and convenience. Moreover, nowadays, mobile phones have been used widely for many purposes and many applications. Therefore, we propose the efficient method enabling to reduce the communication times and route data storage size which are the main problems in thin client model.

3.5.1 Basic concepts in thin client model

When MO begins its trip, it sends the message to the server with its current location. At once the server received the starting trip message from MO, it starts to setup the thread for the monitoring of MO. The server checks the route history that corresponds with the usually used routes, the speed and the direction of MO, and then it predicts the position of MO and offers the predicted route for MO. The predicted position is compared with the actual position and the calculation

of the difference between them is also required. If the difference exceeds the predefined threshold value (θ_D), it is regarded as deviation from a predicted route, the server requests the current position of MO, corrects the parameters of MO and continues the monitoring. At the same time, the server informs that information to LBS server for the requirements of the requested application. As recall from the rich client model, usually large (θ_D) value is defined, e.g., 1km. If the application program requests a more precise location, the server sends the message to MO to decrease (θ_D), it is regarded as deviation from a predicted route, the server requests the current position of MO, corrects the parameters of MO and continues the monitoring. At the same time, the server informs that information to LBS server for the requirements of the requested application. As recall from the rich client model, usually large (θ_D), it is regarded as deviation from a predicted route, the server requests the current position of MO, corrects the parameters of MO and continues the monitoring. At the same time, the server informs that information to LBS server for the requirements of the requested application. As recall from the rich client model, usually large (θ_D) value. This LOD (Level of Detail) control is used in this thin client model as well.

The server extracts the information of usually used routes of MO from the historical data, and offers the predicted route to MO. However, MO may pass through a route which does not exist in route history, i.e; MO uses this route for the first time. In this condition, since there is no route history, the predicted position and the predicted route is done by DRRN (Dead Reckoning on Road Network) method. The server has been updated the route history by adding this new route information for the future use. Therefore, the high accuracy monitoring can be expected just using a mobile phone for communication with the server.

3.5.2 Server side Algorithm for thin client

A server monitors the individual MOs concurrently. Moreover, it offers the predicted route to the MO and submits the information of MO to the application for

the requirement of LBS requests. There are three kinds of modules consist in server side algorithm, communication module, tracking module and API module. Communication module is aiming for communication with MO and LBS application. Tracking module is purposing for tracking individual MO and retrieving the predicted route. API module is planning to answer an inquiry from LBS application. The details of tracking module and API module are shown in next section.

Communication Module for thin client

The algorithm of communication module is defined next and explained in the following.

Algorithm 3 Communication Module algorithm

```
1: repeat
2:   if receiving message from MO then
3:     {code, args} ← receive message from MO
4:     if code = 0 then
5:       Create a new thread to monitoring with args
6:       Inform car# to API module
7:     else if  then
8:       Send args to corresponding thread
9:       Inform to API module
10:    end if
11:  end if
12:  if receiving message from API module then
13:    {LoDinfo, MOlist} ← receive message from API module
14:    Send LoDinfo to designated MOs
15:  end if
16:  if receiving message from thread then
17:    {path, car#} ← receive message from thread
18:    Inform to API module
19:  end if
20: until true
```

When receiving message from MO, the communication module verifies the communication code and the position information of MO (Line 2-3). If communication code is 0, it sets up the tracking module to create a new thread to monitoring the

related MO. Moreover, it sends the individual ID of MO to API module to inform that MO begins its trip (Line 4-6). Otherwise, the communication module sends the position information of MO to the corresponding thread and notifies the API module that MO is deviated from the prediction route (Line 8-9). (Line 12-14) will be taken when the communication module receives LoD information from API module. Next, it transmits LoD information to the designated MOs which specified with MOlist. When receiving a message from the tracking module, the communication module informs a new predicted route information to API module (Line 16-18).

Tracking Module for thin client

This section will describe the server side processing of individual MO tracking module. The op-code using for communication module between MO and server is shown in Table 3.3.

Table 3.3: Communication Module Recieving Codes

code	contents	parameters
0	Starting trip	MO ID, Position Information
1	Synchronization with the predicted position	MO ID, Position Information
2	Deviation from the predicted route	MO ID, Position Information, and the Movement direction
3	Stopping temporary	MO ID, Position Information
4	Restarting trip	MO ID, Position Information
5	Ending trip	MO ID and position information

When MO starts its trip, it sends Code 0 to the server that including individual ID and the current location of MO. After receiving the signal, the server set up the thread for monitoring and loading the corresponding route history. In addition, sever offers the predicted route upon the requested destination and the historical data. If there is no historical data, server will retrieve the predicted route using

DRRN. Code 1 is the signals for communication that occurs when the difference between the predicted position and the actual position getting from GPS receiver exceeds the permissible error. This signal consists of moving object ID and the current location. At this time, server corrects the parameter of MO and synchronizes with the current position of MO. If MO deviates from the predicted route, it transmits Code 2 to the server with moving object ID, current location and the movement direction. According the receiving data, server attempts to predict the new route by the map-matching and sends the new predicted route to MO. When MO stops temporary because of the traffic points, the construction areas or the traffic jams etc., MO sends Code 4 to the server with its ID and current position. By that time, server suspends the monitoring until it receives the restarting trip message from MO (Code 5). If MO reaches its destination, it will generate Code 5 to server with its ID and current location, to inform the ending of its trip. Following this signal, server updates the route history of MO, stops the monitoring and terminates the thread.

Algorithm 5 shows the pseudo code of tracking module. This algorithm is initialized with pos (current position of MO acquires from GPS receiver), car# (MO ID) and BP# (ID of starting BP). Server verifies the FUR from the historical data of MO, retrieves the predicted route and set it in rt. After that server informs the designated MO ID and the predicted route to communication module (Line1-3). As described before, server synchronizes the position and predicts the new route if late arrival, early arrival or deviation from the predicted route based on the op-code (1-4) of communication module (Line 4-19). Server will stop the monitoring when receives op-code (5) and terminate the thread (Line 20).

API Module

API module has aimed to offer the MO information to LBS application and to receive the permissible error value if more precise location is required. The op-code using in API module is illustrated in Table 3.4.

Algorithm 4 Individual tracking module algorithm

Require: *pos*,*car#* and *BP#*

1: Read the FUR of the car# according starting BP#
2: Retrieve the most usually used route from pos in FUR, and then set the result to rt.
3: Inform $\{car\#, rt\}$ to communication module
4: mode $\leftarrow$ moving
5: **repeat**
6: $\{code, args\} \leftarrow$ receive from communication module
7: **if** $code = 1$ **then**
8: $pos \leftarrow$ convert $args$ into position information
9: **else if** $code = 2$ **then**
10: $\{pos, dir\} \leftarrow$ convert $args$ into position information and direction
11: Retrieve the most usually used route from pos and dir in FUR, then set the result to rt.
12: Inform $\{car\#, rt\}$ to communication module
13: **else if** $code = 3$ **then**
14: $pos \leftarrow$ convert $args$ into position information
15: $mode \leftarrow$ suspend
16: **else if** $code = 4$ **then**
17: $pos \leftarrow$ convert $args$ into position information
18: $mode \leftarrow$ moving
19: **end if**
20: **until** $code = 5$
21: Exit Thread

Algorithm (2) describes the pseudo code for API module. (Line 1-3) corresponds to acquire the request message from LBS application including API module op-code and MO information (MO ID and position). If LBS application requires a more precise location, Code 0 has been transmitted. At this time, server receives the range of the monitoring area and LoD information with threadshold value which determined by LBS application (Line 4-5). Server investigates the list of MOs which are entering into the range of monitoring area, and then informs range, LoD information and MO list to communication module (Line 6-7). When LBS application is necessary to have the MO position, it sends Code 1 to the server with user ID and MO ID. Server replies the current position of MO to the related user (Line 8-10). If a server receives the location update information from communication module, it

Table 3.4: API module receving codes

CODE	Content	Parameter
0	Request to change the permissible error (LoD)	range of monitoring area and the value of permissible error
1	Request to send the MO position	user ID and MO ID

will modify the prediction with this information (Line 13-15).

Algorithm 5 API module

```
1: repeat
2:   if receiving message from LBS then
3:     {code, args} ← receive message from LBS
4:     if code = 0 then
5:       {range, LoDinfo} ← convert args into range of application and LoD
         information
6:       MOlist ← find all MO which has possibility of enter range
7:       Inform range, LoDinfo and MOlist to Communication module
8:     else if code = 1 then
9:       {user#, car#} ← convert args into user ID and car ID
10:      Send current position of car# to user#
11:    end if
12:  end if
13:  if receiving message from communication module then
14:    Modify prediction
15:  end if
16: until true
```

3.5.3 Moving object side algorithm for thin client

Moving object has been matched its position acquired by GPS receiver to the road map, and it has a restraint to move along on the specified route which sent from a server. When delay or advance or deviate from the predicted route, or if exceeds the predefined permissible error, MO sends the signal to the server as shown in Table 3.3.

As for the MO in thin client model, the road map of the route is accepted from the server. Moreover, MO also received the related request for the permissible error. This request is due to the requirements from the real-time LBS application and it is necessary to decrease the value of permissible error if MO is in a special area of the town or nearness of the specified position. Table 3.5 shows the op-code that using when MO receives the new predicted route or the LoD information. Table 3.6 denotes the structure of the predicted route. That is composed with the number of nodes (n) of the predicted route and the (x,y) coordinates of node 0 to node (n-1). The storage capacity for the predicted route is directly related to the number of nodes in 8(n)+2. Therefore, the proposed method attempts to decrease the transmission cost and data storage capacity by reducing the number of accessing nodes in the predicted route. The details of nodes compression will be explained in next session.

Table 3.5: Content of client receiving code

Code	Content	Parameter
0	New predicted route	structure of the predicted route
1	Request to change the permissible error (LoD)	range of monitoring area and the value of permissible error

Table 3.6: Structure of the predicted route

n	Number of nodes (2B)
x_0	x coordinates of node 0 (4B and unit of ms)
y_0	y coordinates of node 0 (4B and unit of ms)
...	...
x_{n-1}	x coordinates of node n-1 (4B)
y_{n-1}	y coordinates of node n-1 (4B)
...	...
Range	Total range
Capacity	8n+2

Algorithm 6 Moving Objects (Thin Client) Algorithm

Require: $car\#$,θ_D
1: $\{pos, dir\}$ ←getPosition()
2: send$(0, car\#, pos)$, $synpos \leftarrow pos$
3: dp ←true , $mode$ ←moving
4: **repeat**
5: **if** received code **then**
6: $\{code, args\}$ ←receive()
7: **if** $code = 0$ **then**
8: $rt \leftarrow args$, dp ←false
9: **else if** $code = 1$ **then**
10: $LoD\_info \leftarrow args$
11: **end if**
12: **end if**
13: **if** dp =false **then**
14: $\{pos, dir\}$ ←getPosition()
15: **if** OnRoute(pos, rt) **then**
16: d ←distNow$(pos, synpos, rt, mode)$
17: **if** $d > \theta_D$ **then**
18: **if** $mode$=suspend **then**
19: send$(4, car\#, pos)$, $mode$ ←moving , $synpos \leftarrow pos$
20: **else**
21: **if** distNow$(pos, synpos, rt$,true$) < \theta_D$ **then**
22: send$(3, car\#, pos)$, $mode$ ←suspend, $synpos \leftarrow pos$
23: **else**
24: send$(1, car\#, pos)$, $synpos \leftarrow pos$
25: **end if**
26: **end if**
27: **end if**
28: **else**
29: send$(2, car\#, pos, dir)$, dp ←false , $synpos \leftarrow pos$
30: **end if**
31: **end if**
32: **until** end of trip
33: send$(5, car\#)$

Algorithm 7 illustrates the process executed on thin client side.

When starting a trip, MO acquires its position by GPS receiver and informs to

the server using code 0 from Table 2. Server receives the individual ID and current position of MO, and initializes the thread (Line1-2). MO switches to the waiting state until a new predicted route is received (Line 3).

The following procedure (Line 4-33) is repeated until the end of the trip of MO. MO will receive either the route information or LoD information from a server, and it is described in (Line 5-11) of Algorithm 3. As defined in Table 4, MO obtains a new predicted route from a server if the receiving code is 0. MO releases its waiting state after collecting the route information (Line 7-8). If the receiving code is 1, MO seizes the LoD request with the value of permissible error (Line 9-10). (Line 13-31) denoted the processing when MO arrives early or late, or deviates from the predicted route. Meanwhile, MO acquires its current position by GPS receiver(Line 14).

If the current position of MO is located on the predicted route, (Line15-27) will be processing, and this part is necessary to obtain more accurate monitoring. Server collects the position information of MO and determines the difference between the actual position and the predicted position. (Line 17-27) will be manipulated if the difference exceeds the predefined threshold value (θ_D). At this situation, server will synchronize the current position and the predicted position of MO (Line 24). To maintain the accuracy of monitoring, if MO stops temporarily, MO informs its current position to a server for position synchronization (Line 21-22). Similarly, starting again its trip, MO sends its current position to the server for the same process (Line 18-19).

On the other hand, if MO deviates from the predicted route (i.e. the current position of MO is not located on the predicted route), (Line 28-29) will be executed for a new predicted route.

The above processes are repeated until MO informs to a server about the end of its trip (Line 33).

3.5.4 Minimization of data storage capacity and communication cost

The server processing in the thin client model is similar to the rich client model. However, in thin client model, MO unable to predict the route and it is always necessary to obtain the route information from a server. As a result, the communication times will increase in thin client model and it will cause the low scalability of the system. This section evaluates the method to reduce the communication cost between MO and a server which is the main problem in thin client model.

Generally, the road segment that connects the intersections (i.e. nodes) is defined as a straight line and it is expressed to the road shape on the map. Accordingly, the predicted route is also expressed by two or more nodes. Therefore, the capacity of route information sent to MO can be reduced by minimizing the number of nodes which including in the predicted route. The reduction of nodes number will be simplified by selecting the node which has the error margin exceeds the length of predefined threshold value ($dthr$). It will be explained the details in the following section.

For example, Fig. 3.5 is represented as a predicted route segment which composed with the number of six nodes. First of all, selecting only the start point and the end point of the route and assume that the straight line connecting between them. The selecting node will be indicated in red and the others will be in blue. And then calculating the distance between the rest of nodes and the straight line connecting the start and end point. That distance is defined as the error margin of the individual node and the maximum distance among them is denoted as $dmax$. All of these steps are described in Fig. 3.6. The process will be continue with the comparison of $dthr$ and $dmax$. If the node that has the value of $dmax$ outreaches the predefined $dthr$, that note is selected and added to the simplification of the road. After that selecting again the maximum error margin and compare with $dthr$ until there is no more $dmax$ exceeds $dthr$. Fig. 3.7 summarizes the steps for selecting

the nodes.

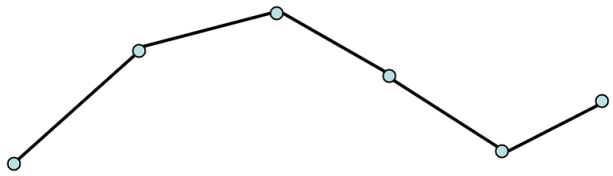


Figure 3.5: Nodes of the predicted route segment

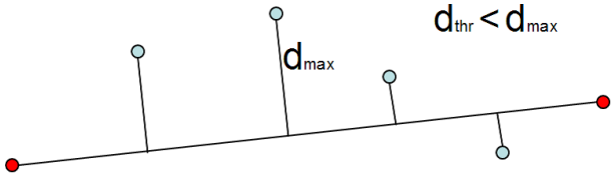


Figure 3.6: Define and choose the maximum error margin

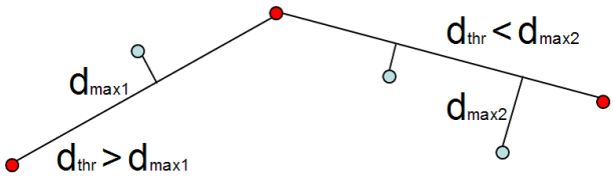


Figure 3.7: Process of selecting nodes added to the predicted route

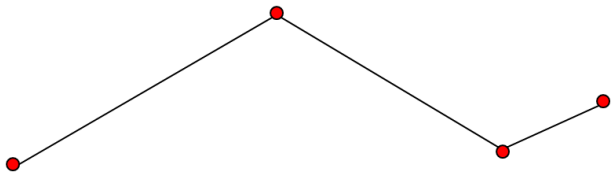


Figure 3.8: Simplification of node reduction

Finally, the road segment is simplified with the reducing number of nodes as shown in Fig. 3.8. When permissible error *dthr* set to 10m, and applied to the actual predicted route, data storage capacity will be able to reduce to one fourth of the original storage approximately by this simplification.

3.6 Experimental results of thin client model

3.6.1 Comparison of communication cost

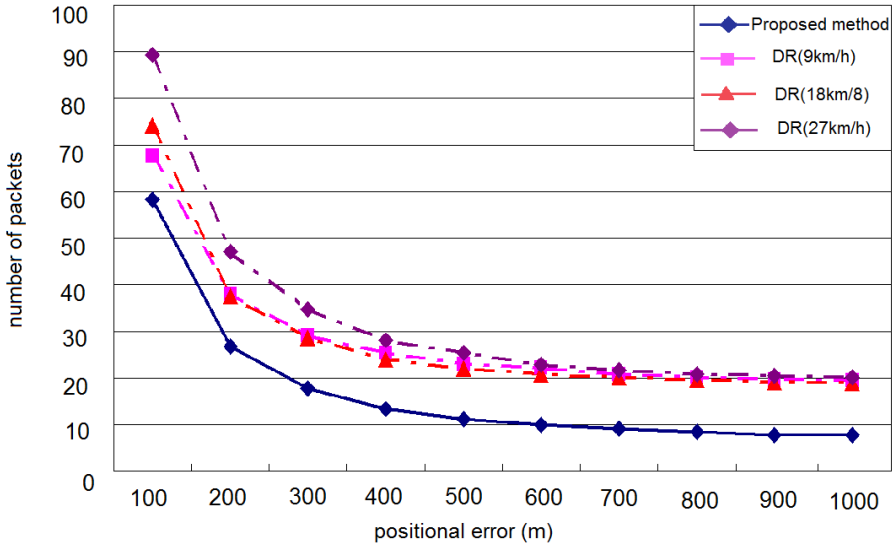


Figure 3.9: Comparison of communication cost

As in thin client model, a similar experiment was done in thin client model. Fig shows the result of the experiment. Here also, the communication cost of the proposed method was compared with that using dead-reckoning on road network (DRRN) for the entire trip to evaluate the efficiency of the proposed method. During experiment, it was assumed that one communication requires only one packet, and therefore, the efficiency was evaluated using the number of packets sent. And the MO's speed was varied from 9 to 27 km/h in increments of 9 km/h.

The horizontal axis in Fig. 3.9 is the permissible positional error (θ_D), and

the vertical axis is the average number of communication packets needed to keep tracking under a θ_D value from 100 m to 1 km. As in both experiments, though the best result in thin client was obtained when the speed was set 9 or 18 km/h, the packet number with dead-reckoning is not so sensitive to the MO's speed, especially when the θ_D value is large. This is because most packets in DRRN are only used to report the deviations of the route (i.e. the road changes).

3.6.2 Summary

Here, we proposed a method for the real-time monitoring of MOs on a road network using FURs. Because conventional real-time monitoring does not use knowledge about routes, scalability remains low and the accuracy of tracking is also low. On the other hand, using FURs extracted from historical trajectories decreases the communication cost and achieves highly accurate monitoring. Our proposed method outperforms the conventional method DRRN by factors from 2 to 4 in thick client and 4 to 6 in thin client.

CHAPTER 4

Shortest Path Finder With Light Materialized Path View for LBS

With the popularity of location based services (LBS), most existing work on path computation has been mainly focused on the shortest path search problem. Therefore, fast shortest path search between two points on a road network is essential demand in location based services (LBS). For this purpose, Dijkstra's algorithm, A* algorithm, and several types of road network distance materialization methods have been studied. The distance materialization approach is faster than the former two algorithms, however, it results in a huge amount of data. Besides, it requires a huge amount of operations when a change is occurred on the road network by, for example, traffic accident or road maintenance. In this thesis, we propose a shortest path search algorithm based on materialized-path-view constructed only on partitioned subgraphs, and its three variations referring different levels of distance materialization. A road network is partitioned into the subgraphs, and the distance materialization is performed only in the subgraphs. Therefore, the amount of pre-computed data is greatly reduced. The shortest path is retrieved by a best-first-search using a priority queue. The difference between three variations

of the algorithm is the materialization level of the distance in the subgraphs. The performance of them is evaluated comparing with A* algorithm and HEPV experimentally. In this thesis, through the results, we show that our proposed algorithm outperforms the conventional methods.

4.1 Preliminaries

Point of interest (POI) queries based on the road network distance become an important role on location based services (LBS). For example, queries to find the nearest neighbor POIs to a specified query point (kNN query), and to find all POIs within a specified distance from a query point (range query). For these queries, the optimization on the distance or the time of travel along the road network is important besides the Euclidean distance.

A shortest path query finds the shortest distance route between specified two points (s and d) on a road network. For this purpose, Dijkstra's algorithm and A* algorithm have been used. These algorithms refer an adjacency list to find the neighboring nodes to a currently noticed node. When two specified points (s and d) are located on a long distance, they need much repetitive processing (*node-expansions*). Therefore, the processing time increases rapidly in accordance with the length of the shortest path.

Several methods based on materialized path view (MPV) have also been proposed for the fast road network distance computation. They retrieve the distance by looking up a pre-computed distance table. When two points are located on the road network nodes, the distance can be obtained by only one access to the table. Generally, two points are not always located on nodes, therefore, at most 4 times access is required. In any case, the road network distance can be determined in a constant time by using the MPV.

However, this MPV has the following problems: (1) Usually, a road network contains a large amount of nodes, and the data size of the MPV is proportional

to the square of the number of nodes. Therefore, the data amount of the distance table becomes huge for a large size of the road network. (2) Very long processing time is necessary to construct MPV table, because the distance must be calculated over all combinations of node pairs. As concerns to the data amount of the table, when the total number of nodes in a graph is $1,000,000$ (it corresponds to a road network over the range about 100km square), the number of elements in MPV table becomes 10^{12}, therefore several TB memory is required. (3) When the weight values (e.g. length) of some links in the network are changed by a traffic accident or a construction, these changes affect the wide area on the table. This update also requires a long processing time.

To cope with these problems, hierarchical MPV methods have been proposed. These methods alleviate the problems described above, however, the problems cannot be avoided authentically. A change in a leaf level affects to the upper levels. Long computation time is necessary for the upper level distance calculation. The data amount in a high level layer is not always smaller than that of the leaf level, in opposition to a usual hierarchical tree structure.

Car navigation systems sometimes search the shortest paths between two points located very far away. In this situation, the most suitable search method can be considered as a hierarchical structure based on the types of roads [102]. For example, roads are divided into the highway and the usual road. First, we search a rough shortest path on the highway network, and then search the path between each given terminal point and the access point of the highway on the usual road network. Though this method may not give the shortest path, the result is adequate for the usual purpose.

In a query for LBS, on the other hand, the shortest path must be determined from a large number of candidates, and the area where candidates exist is limited in a confined area, for example, the searching in an area having 50km radius centered the query point. Moreover, point of interests (POIs) as query targets are usually

located on the usual road network. Therefore, it is not suitable to adopt the method based on the road attribute hierarchy to LBS.

In this thesis, we propose a shortest path search algorithm based on a lightweight local distance materialization, which is constructed on a partition of a road network. These methods outperform A* algorithm, and they reduce the data amount drastically comparing with the conventional hierarchical distance materialization methods. And then, the proposed methods and the conventional methods are evaluated experimentally.

4.2 Shortest Path Finder

4.2.1 Data structure

A road network is modeled as a directed graph $G(V, E, W)$, where V is a set of nodes (intersections), E is the set of edges (road segments), and W is the set of link weights. A fragment $SG_i(V_i, E_i, W_i)$ of a graph $G(V, E, W)$ is a partitioned subgraph, where $V_i \in V$, $E_i \in E$, and $W_i \in W$. If the end points of an edge $e_{jk} \in E_i$ are v_j and v_k, then $v_j \in V_i$ and $v_k \in V_i$. This subgraph is denoted as SG_i starting from here.

Fig. 4.1(a) shows an example of a road network graph, here small circles are nodes and lines are edges. Fig. 4.1(b) depicts a partition of the graph shown Fig. 4.1(a). In this partition, the nodes shown by full-colored inside small circles belong to at least two neighboring subgraphs; i.e., the nodes belonged to the plural subgraphs are called the *border nodes*. Two subgraphs are defined adjacent if they have at least one common border node. The set of border nodes of SG_i is denoted by BV_i. In this partition, each edge belongs to only one subgraph. The nodes shown in white circles in the figure are referred as *inner nodes*: they are the rest of the nodes in a subgraph except the border nodes.

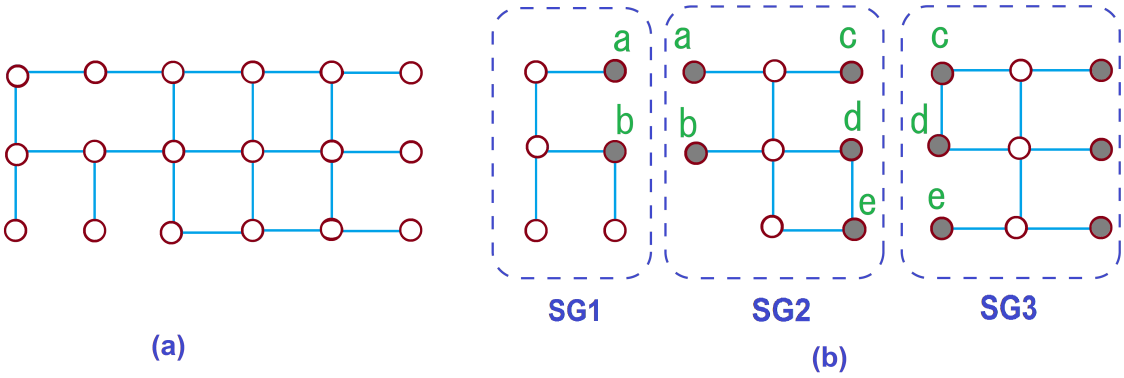


Figure 4.1: Partitioning on flat graph

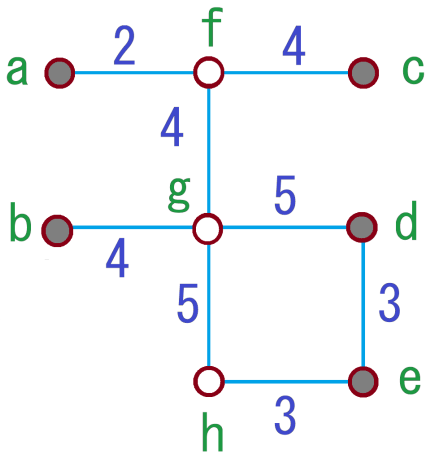


Figure 4.2: Extraction of SG2

Fig. 4.2 extracts SG_2 from Fig. 4.1. The numerical value attached each link shows the weight of the link, for example, the length of the link or the travel time to pass through the link. In our proposed method, we assume the weight as the length of the link.

(a) Border-to-border node distance table

	a	b	c	d	e
a	0	10	6	11	14
b	10	0	12	9	12
c	6	12	0	13	16
d	11	9	13	0	3
e	14	12	16	3	0

(b) Inner-to-border node distance table

	a	b	c	d	e
f	2	8	4	9	12
g	6	4	8	5	8
h	11	9	13	10	3

(c) Node-to-node distance table

	a	b	c	d	e	f	g	h
a	0	10	6	11	14	2	6	11
b	10	0	12	9	12	8	4	9
c	6	12	0	13	16	4	8	13
d	11	9	13	0	3	9	5	10
e	14	12	16	3	0	12	8	3
f	2	8	4	9	12	0	4	9
g	6	4	8	5	8	4	0	5
h	11	9	13	10	3	9	5	0

Figure 4.3: Distance tables used for SG2

Fig. 4.3(a) shows the shortest path length between every two border nodes in SG_2. The lengths are calculated by traveling inside of the subgraph, therefore these values are not always global shortest path lengths. If there is no connected path between a paired nodes inside the subgraph, the infinity value is assigned to the related element of the table. Though the matrix is symmetry in this example, it is not always symmetrical in the real road network because of the existence of one

way road. In the rest of the paper, we refer this table as a border-to-border distance table (BBDT).

Fig. 4.3(b) shows another table, the inner-to-border node distance table (IBDT), which shows the distance from an inner node to a border node. This table is used to retrieve the distance from the starting point as an inner node to a border node. Since the distance on the road network is not symmetric, the transposed matrix of Fig. 4.3(b) is also necessary to obtain the distance between a border node and the destination point.

Fig. 4.3(c) shows the node-to-node distance table (NNDT), listed distances of all combinations of the nodes in SG_2. This table is used to know the distance between two arbitrarily specified nodes. Either IBDT or NNDT is used alternatively in the SPF algorithms described in next section.

4.2.2 Simple Path Finder Algorithm

Fig. 4.4 to Fig. 4.7 shows the processing flow of the shortest path finder (SPF) algorithm in each state. In the following description, s and d denote the starting point and the destination point of the shortest path to be retrieved. The SPF is controlled by a best-first search using a priority queue (PQ).

The PQ manages the records constructed by the following items.

$$< p, Cost, dfs, fSG, phase >$$

Here, p is the currently noticed point; s, d, or a border node. $Cost$ is the lower bound road network distance between s and d. The PQ returns the record by ascending order of this value. dfs (distance-from-source) is the shortest road network distance between s and the currently noticed node p.

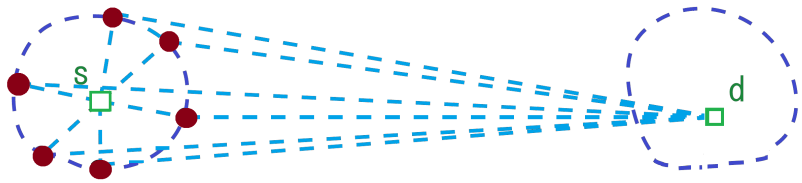


Figure 4.4: Initial state (PHASE0)

fSG is the subgraph ID in which p belongs. The last item, *phase* is a value to show the progress of the processing. It is changed from PHASE0 (initial state) to PHASE3 (final state) according to the progress of the processing.

At first, the subgraph, SG_s, which contains the road segment under s, is determined. Next, $Cost$ is calculated by the equation, $Cost = d_E(s, b_i) + d_E(b_i, d)$, for all border nodes $b_i \in BV_s$ of SG_s. Here, $d_E(x, y)$ denotes the Euclidean distance between x and y. In this initial stage, the following records are composed and enqueued to the PQ. In this processing stage, the records have $PHASE0$ as the *phase* value and the processing state is shown in Fig. 4.4.

$$< b_i, d_E(s, b_i) + d_E(b_i, d), 0, SG_s, PHASE0 > \quad for\ all\ b_i \in BV_s$$

Next, a record (e) that has minimum $Cost$ value is dequeued from the PQ as shown in Fig. 4.5. At the beginning of the processing, $e.phase$ is $PHASE0$.

Figure 4.5: Changing state (PHASE0 to PHASE1)

For the border node $e.p$, the road network distance $d_N(s, e.p)$ is calculated. Here, $d_N(x, y)$ denotes the road network distance between x and y. The way to determine the road network distance is described in Sect. 4.2.3. $Cost$ value for this node is calculated by the equation $Cost = d_N(s, e.p) + d_E(e.p, d)$, composing the following record, and then it is enqueued in the PQ.

$$< e.p, Cost, d_N(s, e.p), e.fSG, PHASE1 >$$

In Fig. 4.6, when the *phase* value of the obtained record (e) from the PQ is $PHASE1$, the road network distance from s to the current node ($e.p$) has already been determined. All subgraphs that contain $e.p$ as a border node is also determined. And then, for each subgraph SG_n, $Cost$ is calculated by the following equation.

$$Cost = e.dfs + d_N(p, b_i) + d_E(b_i, d) \quad (b_i \in BV_n)$$

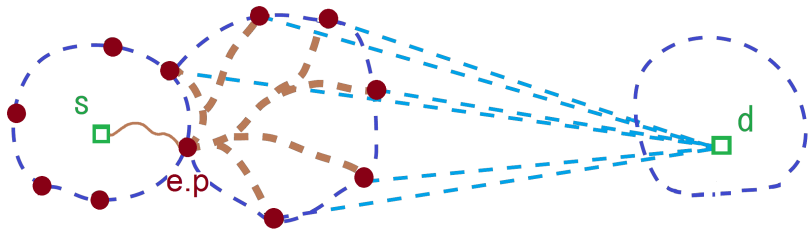


Figure 4.6: Processing in neighboring subgraph (PHASE1)

Here, BV_n is a border node set of SG_n. The following record is composed, and then it is enqueued in the PQ.

$$< b_i, Cost, e.dfs + d_N(p, b_i), SG_n, PHASE1 >$$

Continuing the processing, when a record obtained from the PQ reaches a border node of the subgraph containing d, the record shown below is composed and it is enqueued in the PQ.

$$< e.p, e.dfs + d_E(e.p, d), e.dfs, e.fSG, PHASE2 >$$

In this case, the value of the road network distance from s to $e.p$ plus the Euclidean distance between $e.p$ and d is assigned to $Cost$ value, and $PHASE2$ is assigned to *phase* value.

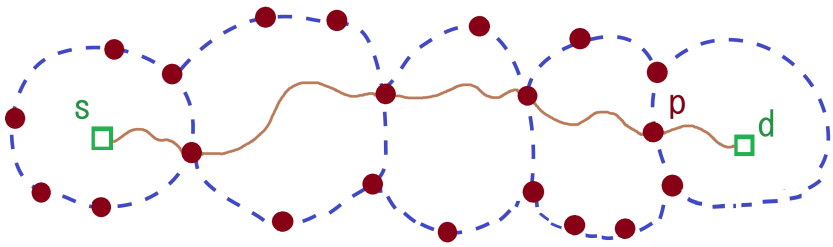


Figure 4.7: Processing states (PHASE2 to PHASE3)

When the *phase* value of the dequeued record from the PQ is $PHASE2$, the road network distance between $e.p$ and d is calculated. Composing the following record, and it is enqueued into the PQ.

$$< e.p, e.dfs + d_N(e.p, d), e.dfs, SG_d, PHASE3 >$$

The way how to determine $d_N(e.p, d)$ is described in next session.

When the *phase* value of the dequeued record is $PHASE3$, the shortest path distance between s and d has been determined as shown in Fig. 4.7. The fact that the record is dequeued from the PQ means it has the minimum *Cost* value among all records contained in the PQ. It means the shortest path distance is determined. Therefore, return the distance, and then the searching process is terminated.

Algorithm 7 shows the pseudo-code of the described procedure above.

Algorithm 7 SPF: simple path finder

Require: s,d

1: $SG_s \leftarrow determineCell(s)$
2: $SG_d \leftarrow determineCell(d)$
3: $CS \leftarrow \emptyset$
4: **for all** $p \in BN.SG_s$ **do**
5: $PQ.enqueue(p, d_E(s,p) + d_E(p,e), 0, SG_s, PHASE0)$
6: **end for**
7: **while** PQ is not empty **do**
8: $e \leftarrow PQ.dequeue()$
9: **if** $e.phase = PHASE3$ **then**
10: **break**
11: **else if** $e.phase = PHASE0$ **then**
12: $c \leftarrow d_N(s, e.p) + d_E(e.p, d)$
13: $PQ.enqueue(e.p, c, d_N(s, e.p), SG_s, PHASE1)$
14: **else if** $e.phase = PAHSE2$ **then**
15: $c \leftarrow e.dfs + d_N(e.p, d)$
16: $PQ.enqueue(e.p, c, c, SG_d, PHASE3)$
17: **else**
18: $nCell \leftarrow e.p$
19: **for all** $cell \in nCell$ **do**
20: **for all** $b \in BN.cell$ **do**
21: $c \leftarrow e.dfs + d_N(e.p, b) + d_E(b, d)$
22: $PQ.enqueue(e.p, c, e.dfs + d_N(e.p, b), cell, PHASE1)$
23: **end for**
24: **end for**
25: **end if**
26: **end while**
27: $path \leftarrow calculatePATH(s, d, CS)$
28: RETURN $path$

4.2.3 Distance calculation method inside subgraph

This section describes three variations of SPF; SPFLM (SPF with light materialization), SPFMM (SPF with medium materialization), and SPFFM (SPF with full materialization). The difference between these methods is how to determine the distance between one of the specified points (s and d) and the border nodes of the subgraph where the point belongs to.

SPFLM calculates the distance by A* algorithm referring usual adjacency list of the road network. Usual A* algorithm, we hereafter refer this as pair-wise A* (PWA*) algorithm, can search the shortest path efficiently when two terminal points are located nearly. The extent of the subgraph is small, hence, the distance determination inside a subgraph satisfies this condition. However, this operation is invoked several times in transition from PHASE0 to PHASE1 and from PHASE2 to PHASE3.

When the target border nodes are located near each other, several nodes are expanded multiple times. As a consequence, the total processing time of SPFLM increases. To avoid this, we adopt SSMTA* algorithm [103] that guarantees the number of times of node-expansions only once for the same nodes during the shortest paths searching for a set of target points. Though, this algorithm was proposed to search multiple target points simultaneously, the target point in SPF algorithm is always added one by one.

When there is no route between n_1 and n_2, the whole nodes in the subgraph that belongs to both n_1 and n_2 are expanded. However, all nodes in a subgraph are once expanded, the path distance between n_1 and the arbitrary boundary node in the subgraph can be obtained by retrieving the closed set, CS.

SPFMM obtains the distance between s and a border-node and the distance between a border-node and d by referring the IBDT. However, when s and d are located in the same subgraph, the distance between s and d cannot be obtained by the IBDT: the distance is obtained by PWA* algorithm for this case.

The last algorithm, SPFFM, determines the shortest path from a point to a border node in a subgraph by referring the NNDT, which has all combinations of the distances between any two inner-nodes. The distances in the NNDT are calculated only inside a subgraph, therefore they are not always the global minimum distances. Hence, the shortest path searching by Algorithm 7 is also necessary even when s and d are located in the same subgraph.

To determine the shortest path route between two border nodes, the Next Hop Table (NHT) is used [14]. An element in this table indicates the direct successor node on the shortest path from a current node to the destination node. The number of times to look-up the table to restore the shortest path route is proportional to the number of nodes on the shortest path.

In the above explanation, s and d are located on the network nodes. However, in the usual use, these points are located on the road links. To generalize the suitable situation is easy. When s is located on link e_s and d is located on link e_d respectively, the shortest path length can be determined to apply SPF algorithms for four combinations among both endpoints of e_s and both endpoints of e_d. On this occasion, the minimum of distances from s to d via any endpoint of both links is added to the results.

Table 4.1 shows the reference tables described in previous section using for three SPF algorithms.

Table 4.1: Referring Tables in different SPF algorithms

Data Table	SPFLM	SPFMM	SPFFM
BBDT	●	●	●
IBDT		●	
NNDT			●
Adj. List	●	●	

4.3 Experimental results

This section evaluates the performance of three proposed variations; SPFLM, SPFMM, and SPFFM, by comparison with two representative conventional methods, PWA* algorithm and HEPV. All algorithms are implemented by Java, and are evaluated

on a PC with an Intel Core i7 CPU 960 (3.2GHz), 9GB memory. Table 4.2 describe the road network maps dara used in this experiment, and Table 4.3 shows the data size difference in each map.

Table 4.2: Road network maps data used in the experiments

Map name	# of nodes	# of links	Area size	Adj. List	BBDT	IBDT	NNDT
MapS	16,284	24,914	168 km^2	1.5MB	1.1MB	4.1MB	13.4MB
MapM	109,373	81,233	284 km^2	6.8MB	4.5MB	17.4MB	65.6MB
MapL	465,245	638,282	3,797 km^2	39.7MB	26.1MB	100.8MB	373.8MB

Table 4.3: Data size in each map (MB)

Map name	PWA*	SPFLM	SPFMM	SPFFM	HEPV	Next Hop
MapS	1.5	2.6	6.7	14.5	30.1	13.4
MapM	6.8	11.3	28.7	70.1	376.1	65.6
MapL	39.7	65.8	166.6	400.0	8,287.6	373.8

The adjacency list was prepared as follows: (1) Peano-Hilbert order [104] was assigned to all nodes; (2) neighboring nodes in this order were clustered into 8KB blocks. For the adjacency list management, a 0.5MB (64 block) LRU buffer was assigned. Partitioning of a road network into the subgraphs are performed by the following method: (1) we selected nodes (source nodes) on the given road network for a specified number of divisions: (2) applying multiple sources Dijkstra's algorithm, we categorized each node into a subgraph that has the same source node as

nearest neighbor. Three types of tables, BBDT, IBDT, and NNDT were prepared for each subgraph. Higher level of HEPV is constructed based on this partition.

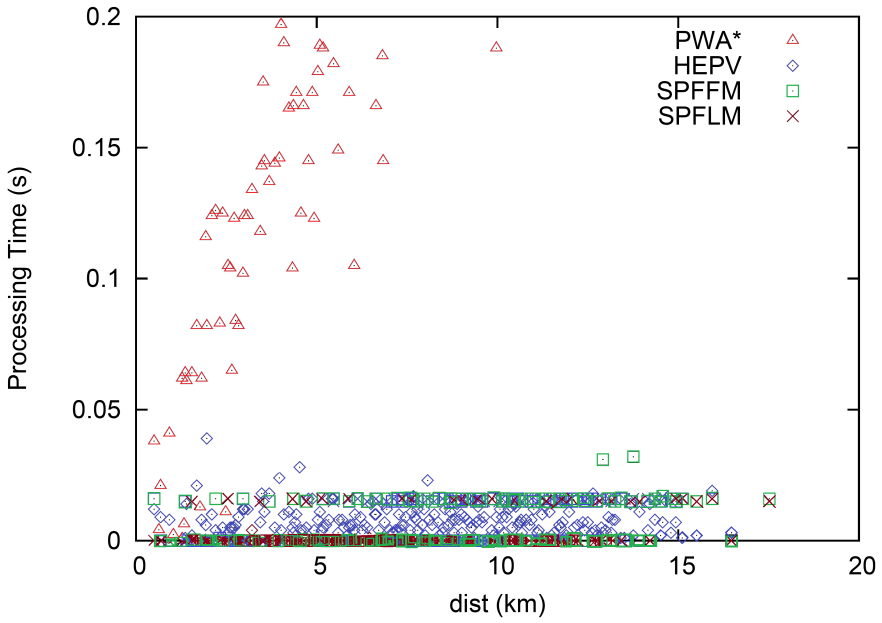


Figure 4.8: Processing time when s and d are placed on nodes (MapS)

Fig. 4.8 compares the processing time of the shortest path searching among the PWA*, SPFLM, SPFFM, and two layered HEPV, using MapS divided into 100 subgraphs. The horizontal axis shows the distance between s and d. We generated 1,000 pairs of s and d by a pseudo-random sequence. For each $s-d$ pair, the shortest path was searched by five algorithms. (SPFMM is omitted from this figure to avoid intricacy: it performed almost the same as SPFFM.) This figure presents the results that is selected one after every 5 queries. All LRU buffers were cleared in advance

for every query. The most of processing time by SPFLM, SPFFM, and HEPV stay under 20 ms over the whole distance range. Meanwhile, the processing time of PWA* increases almost linearly in accordance with the increase of the distance.

Next, we generate several sets of points by a pseudo-random sequence to simulate points of interest (POI) on the road network links. The number of generated points were specified by a probability $Prob$. For example, when $Prob = 0.01$, a POI exists on 100 road links.

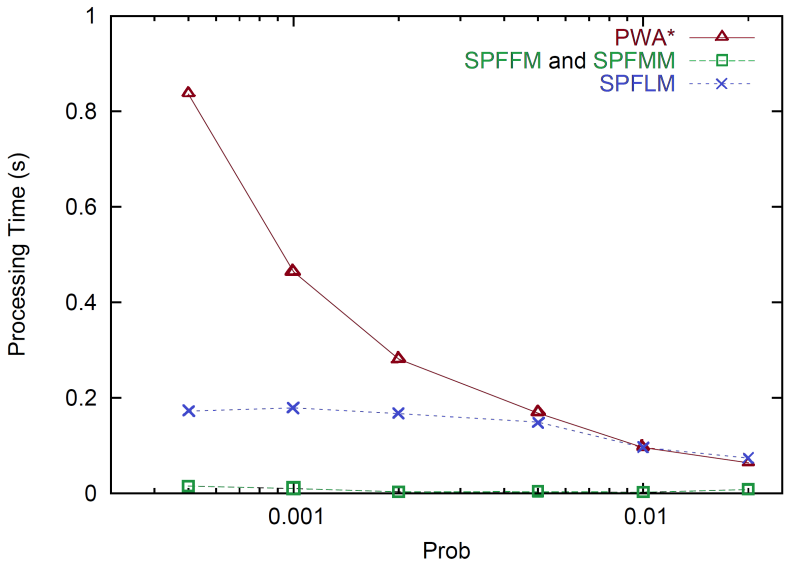


Figure 4.9: Finding 10 shortest paths in MapS

We searched 10 nearest neighbor (NN) POIs of a query point (q) in Euclidian distance. After that, the road network distances are computed for the found POIs by PWA*, SPFLM, SPFMM, and SPFFM. We determined 10 query points randomly on the road network. Fig. 4.9 shows the average processing time spent to determine

10 shortest paths on MapS.

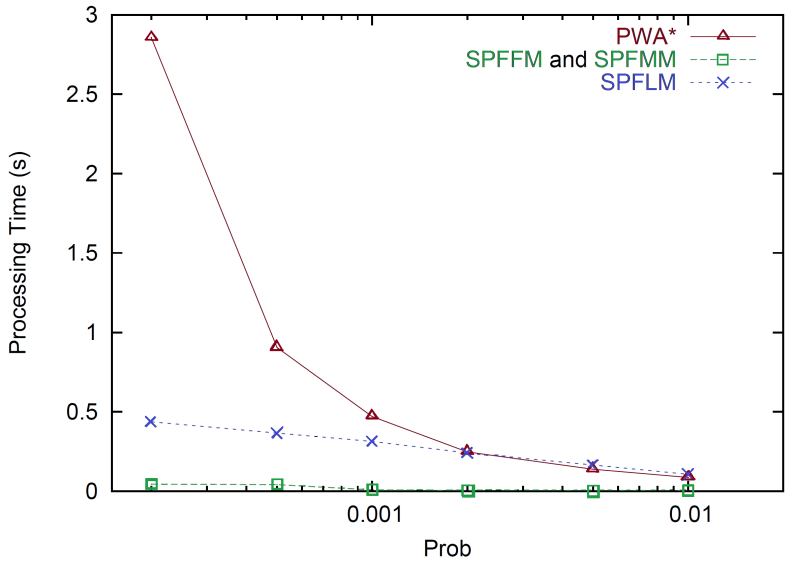


Figure 4.10: Finding 10 shortest paths in MapM

Then Fig. 4.10 and Fig. 4.11 show the results of the same experiments over MapM and MapL, respectively. These three results show similar processing times for the same *Prob*. For denser than $Prob = 0.002$, the processing time of the SPFLM shows almost the same value with PWA*. This is because the path length is small in high *Prob* values, and PWA* algorithm can run fast.

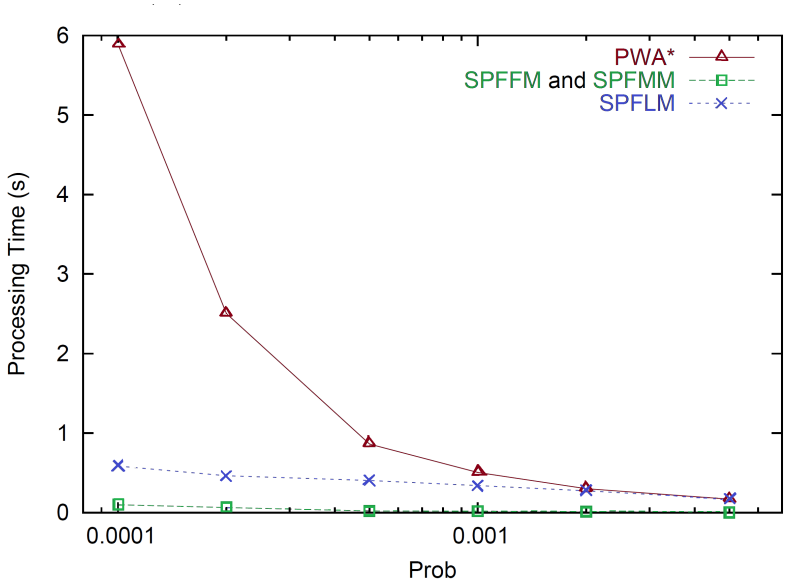


Figure 4.11: Finding 10 shortest paths in MapL

Fig. 4.12 compares the processing time of SPFLM and SPFFM by varying the average number of nodes in subgraphs. In SPFFM, the processing time is minimum when the average number of nodes is 240, On the other hand, in SPFLM, it is minimum when the average is 150: the processing time increases in accordance with the number of nodes. SPFLM needs to search the distance between border nodes of a subgraph, and the distance is calculated by PWA* algorithm. Therefore, when the size of a subgraph is smaller, the processing cost is shorter.

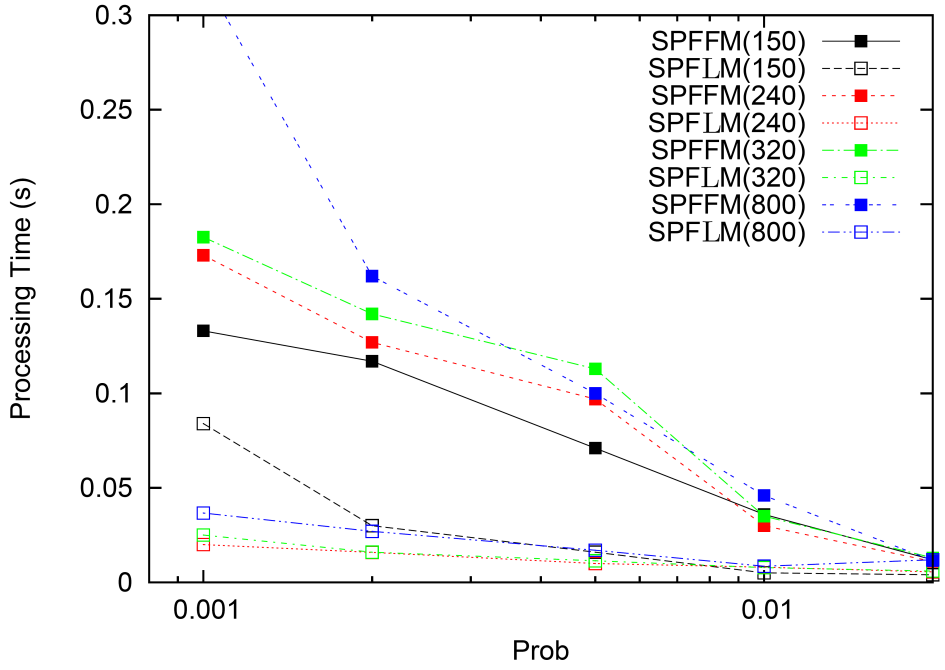


Figure 4.12: Processing time of the routes to 10-NN POIs

4.4 Summary

In this thesis, we propose a shortest path search algorithm and its three variations using the light distance materialization that are suitable for LBS. The data amount used in these methods can be reduced in comparing with the conventional hierarchical network distance materialized methods; HEPV and HiTi. Especially, SPFLM reduces the data amount drastically. On the other hand, SPFMM and SPFFM achieve similar time efficiency with the hierarchical encoded path view.

CHAPTER 5

Efficient Reverse kNN Query Algorithm on Road Network Distances Using Partitioned Subgraphs

Nowadays, with the wide use of location sensing devices (such as GPS receivers), location based services (LBS) are getting popular. The queries related with location information also become an important role in location based services. One such query type is the Reverse k-Nearest-Neighbor (RkNN) query. In this thesis, we propose an efficient RkNN search algorithm using a simple materialized path view (SMPV). In addition, we adopt the incremental Euclidean restriction (IER) strategy for fast kNN queries which is the main function in RkNN query. A Reverse k-Nearest-Neighbor (RkNN) query finds the data points that take the query point as one of their k nearest neighbors. In most conventional algorithms, kNN search on every visited node is needed. This causes a large number of node expansions; therefore, RkNN queries on road network distances require long processing times. The processing time is increased especially when points of interest (POIs) are sparsely distributed. The key idea we have in our solution is this: the SMPV used in our proposed algorithm only constructs an individual partitioned subgraph, therefore the amount of data is drastically reduced in comparison with the con-

ventional MPVs. According to our experimental results using real road network data, our proposed method shows processing time that were 100 times faster than conventional approaches when POIs are sparsely distributed on the road network.

5.1 Preliminaries

A Reverse k-Nearest-Neighbor query finds the data points that are influenced by the query points. It can be applied in LBSs to answer interesting location related questions. Therefore, efficient reverse k-nearest neighbor (RkNN) query algorithms have recently gathered increased attention. This type of query is required in a wide variety of applications, including decision support, facility management, taxi allocation, location-based services, advertisement distribution, and games. For example, in a disaster, a support rescue team may use a RNN query to find other teams for which is the closest team to get support from it.

Depending on the user's requirements, RkNN query may need to retrieve an answer in Euclidean distance or in spatial networks (i.e., a road network); however, most existing algorithms are based on the Euclidean distance. In contrast, in location-based services (LBS) using mobile phones or in-car navigation systems, queries based on road network distances are required. When a river or mountain lies between two points, the road network distance is apt to substantially differ from the Euclidean distance. Therefore, we propose an efficient algorithm to obtain RkNN for road network distances.

When a set of POIs P and query point q (usually with $q \in P$) are given, a RkNN query finds the POIs for which q is included in their kNN; i.e.

$$\mathrm{R}k\mathrm{NN}(q) = \{p \in P | d(p,q) \leq d(p, p_k(p))\}$$

where $p_k(p)$ is the k-th NN of p.

A simple approach to handling the RkNN query is to find kNN for every POI

in P in advance; in doing so, the POIs for which q is included in the result list can be easily found. The cost of this primitive method is bounded by $O(n^2)$ when the number of POI is n. Several algorithms have been proposed to reduce the cost in Euclidean space; however, very limited research has focused on queries involving road network distances, and these methods require long processing times, especially with POIs sparsely distributed on the road network or when k is large.

In this thesis, we propose a fast RkNN query algorithm for road network distances using simple materialized path view (SMPV) data [105]. This algorithm runs on SMPV and refers to the SMPV tables to obtain road network distances for pairs of terminal points. The proposed algorithm searches RkNN approximately 100 times faster than conventional algorithms when POIs are sparely distributed in the road network.

5.2 Basic method for RkNN search

A basic method for RkNN search in road networks, followed by an improved method based on the incremental Euclidean restriction (IER) method, is described in this section. Yiu et al. [73] presented the following lemma for RkNN search in a road network.

Lemma 1. *Let q be a query point, n be a road network node, and p be a POI which satisfies $d_N(q,n) > d_N(p,n)$. For any POI $p'(\neq p)$ whose shortest path to q passes through n, $d_N(q,p') > d_N(p,p')$. That means p' is not a RNN of q.*

This Lemma is proved in [73]. $d_N(a,b)$ denotes the road network distance between a and b.

An example of a simple road network is shown in Fig. 5.1. In this figure, the circles are represented as road network nodes and the squares overlapped on the circles are represented as POIs. We assume POIs are located on nodes; however, this restriction can be easily relaxed [75]. The value attached to edges show the

cost (e.g., distance) of the edge. In this figure, query point q exits on the network node C. When we observe F by query point q, the NN of network node F is data point E. And the NN of E is other data point D; hence, C is not the NN of E. If we substitute n for F, p for E, and p' for D in Lemma 1, we obtain the relations $d_N(C, F) > d_N(E, F)$ and $d_N(C, D) > d_N(E, D)$. Therefore, even if we continue the search beyond F, we cannot find the RNN of q. When the search reaches F, there are two edges connected to F. One edge is connected to E; however, E is not the RNN of C because E has data point D as its NN. On the other edge, passing through network node H, there is POI G. However, G cannot be the RNN of C because the distance from G to E passing through F $(d_N(G, E)viaFis15)$ is at least shorter than the distance from G to C passing through F $(d_N(G, C)viaFis16)$. Therefore, the paths passing through F can be safely pruned from the search.

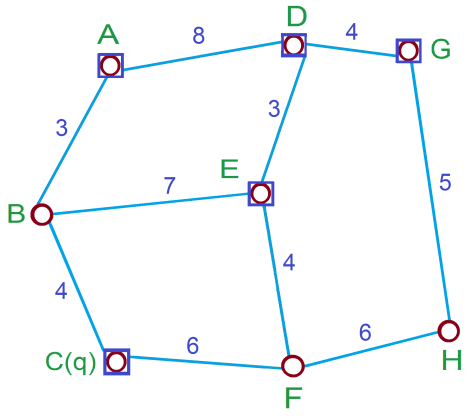


Figure 5.1: Example of a road network

Yie et al. [73] proposed the Eager algorithm based on Lemma 1 and a branch-and-bound approach. The Eager algorithm visits road network nodes from q to surrounding nodes in a method similar to that of Dijkstra's algorithm. When query

q is on C in Fig. 5.1, node B is visited first. Next, at most the number of k NNs of B is searched within the distance, suggesting that the distance between q and B is $D = d_N(B, C)$. This function is called rangeNN(n, q, D). For simplicity, we assume k is 1.

In Fig. 5.1, A is found as B' s NN. Thereafter, we find that the NN of A from the POI set to verify $q(C)$ is the NN of A. This function is called verify(p, k, q) and returns true when q is the NN of p; otherwise, it returns false. In this example, the result of verify$(A, 1, q)$ is true; therefore, A is determined as the RNN of q. The next visited node is F; thus, rangeNN$(F, q, 6)$ is issued and E is obtained as the NN of F. To check whether E is a RNN of q, verify$(E, 1, q)$ is issued; however, false is obtained in this case. Hence, the edges beyond F are safely pruned. At this time, there is no search path left; therefore, the search processing is terminated.

Yiu's Eager algorithm uses Dijkstra's algorithm for verify(p, k, q) and rangeNN (n, q, D). For simplicity, these functions are hereafter denoted as verify and rangeNN. When the density of POIs is high and the search area is small, this algorithm completes quickly. In contrast, when the density is low or k is large, the processing time becomes very long because the search area becomes large.

The inefficiency of the Eager algorithm is summarized as follows:

- A large search area for verify and rangeNN functions and
- a large increase of processing time by performing rangeNN on every visited nodes.

To cope with these problems, we propose a method to adapt an IER framework for the verify and rangeNN functions. Further, in the next section, we propose a very efficient method of RkNN search to perform these queries on SMPV structure.

5.3 RkNN query algorithm on partitioned subgraph

The problems of the Eager algorithm described in earlier section were that (1) the rangeNN and verify functions require long processing times and (2) rangeNN is invoked for every visited road network node. To address these problems, we present the following two proposals: (1) to adapt an IER framework for both rangeNN and verify and (2) to run the Eager algorithm on border nodes in SMPV.

5.3.1 kNN query using an IER framework

Incremental euclidean restriction (IER) framework adaptable for several types of queries on road networks is proposed by Papadias et al. [1]. The basis of this framework is that the Euclidean distance between two terminal points is always a lower bound of the road network distance. For example, when point q and distance r are specified, and the task is to find all POIs to which the distance from q is less than or equal to r, this query (a range query) can be performed via the following two steps:

(a) Search all POIs residing in the circle whose center is q and radius is r and

(b) verify the road network distance of each POI obtained by the above step to eliminate POIs of which road network distances are larger than r.

Queries on Euclidean distance can be performed quickly via a spatial index, e.g., R-trees. In addition, the distance verification can be performed quickly by using the A* algorithm or the algorithm described in Sect. 5.3.3; they are adaptable because two terminal points are given by the query for Euclidean distance.

For the Eager algorithm, IER can be adapted to both rangeNN and verify. First, rangeNN$(n, k, d(n, q))$ is a range query centered at n with range distance $d(n, q)$. This query can be performed by the method mentioned above. The verify(p, k, q) function is essentially a kNN query; therefore, this query can also be efficiently

performed via IER [1]. The merit of using the IER framework increases when k is large and the distribution of POIs is sparse.

5.3.2 RkNN query on an SMPV structure

The most time-consuming step in the Eager algorithm is to perform rangeNN at every expanded node. In the algorithm presented in this subsection, rangeNN is invoked only on the border nodes of the subgraphs to alleviate this problem.

For RkNN search, the same data structure and two distance tables (BBDT and IBDT) for the shortest path finding which described in previous chapter have been used. When query point q is given, the cell in the SMPV structure that q belongs to is determined and the POIs belonging to the subgraph are searched. Let this POI set be P. Next, for each element in P, check whether q is a RkNN or not. This procedure is the same as verify (p, k, q) in the Eager algorithm; verify (p, k, q) searches kNNs of each $p \in P$, and then if q is included in the kNN set, q is decided as a RkNN of p. This check requires a wide-area search and is not exclusive to only a subgraph; it can be efficiently performed using SMPV by IER.

Here, referred Fig. 4.1 from the previous chapter for flat graph and its partitions, Fig. 5.2 is an extraction of SG2 of Fig. 4.1(b). The numerical value attached to each edge shows the weight of the edge. Fig. 5.2 shows the example of a subgraph in which q is a query point and p is a POI included in the subgraph. For simplicity, the following explanation considers the case in which k is 1. By searching for the NN of p, q is obtained as the result. Therefore, p is a RNN of q. Consequently, p is added to the result set.

Next, we enlarge the search area to include the neighboring subgraph. For each border node b_i, the distance from q to b_i is obtained by referencing the IDBT of the subgraph.

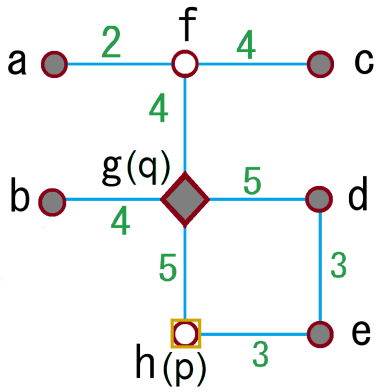


Figure 5.2: Processing in a cell where belonging q

Thereafter, a record is composed and inserted into priority queue PQ; the record is composed as

$$< d, n, p, cid >$$

where d is the road network distance between q and the concerning border node, p is the previous node on the shortest path from q to n, and cid is the cell ID that n belongs to. The first record inserted into PQ is as follows.

$$< d_N(q, b_i), b_i, q, cid >$$

For example, for node a of Fig. 5.2, the record $< 5, d, q, C_{id} >$ is inserted into PQ. The steps described above comprise the StartCell procedure, shown in detail in Algorithm 8.

Algorithm 8 StartCell

```
1: procedureStartCell(q,PQ)
2: CellID ← determineCell(q)
3: P ← findPOIinCell(q)
4: for all p ∈ P do
5:    if verify(p, k, q) then
6:       RS ∪ p
7:    end if
8: end for
9: for all b ∈ BN do
10:   PQ.enQueue(< d_N(q, b), b, q, CellID >)
11: end for
12: endprocedure
```

Next, the RkNN search starts. When a record is dequeued from PQ, the search propagates to the neighboring subgraphs. In Fig. 5.3, subgraph SG2 is the cell in which query point q is included and SG3 is a neighboring subgraph.

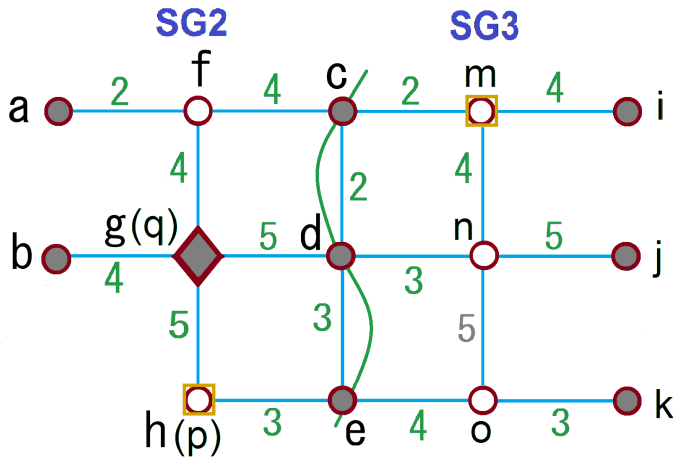


Figure 5.3: Border node expansion

When record r is dequeued from PQ and $r.n$ is the border node d, POIs in SGb are searched. In this subgraph, m is included. Next, the kNNs of m are searched, and if q is included in the kNN set, m is added to the result set. Otherwise, m is ignored. This subgraph can be visited several times from different border nodes. Thereafter, SGb is marked as visited to avoid duplicate searches.

Next, rangeNN is invoked from the border node b_i to find candidate POIs. If the result set is not empty, verify is invoked to check whether each POI is truly an RkNN of q. If the result of verify is true, the POI is added to the result set. If the size of rangeNN is smaller than k, there can exist other RkNNs on the path through this node. Therefore, new records from b_i to the other border nodes in the subgraph are created and inserted into PQ.

Algorithm 9 shows the pseudo-code of the proposed method described above. Lines 3 – 12 are similar to the process described by the Eager algorithm. When the record v is obtained from PQ, at most k NNs of the network node $v.n$ are searched and put into KNN. For each element p of KNN, p is checked whether q is included in its kNN. If so, p is inserted into the result set R.

Algorithm 9 RkNN

```
1: function RkNN(q)
2: PQ ← ∅, R ← ∅
3: StartCell(q, PQ)
4: while PQ is not empty do
5:    v ← PQ.deQueue()
6:    CS.add(v)
7:    KNN ← rangeNN(v.n, k, d_N(v.n, q), PQ)
8:    for all <p in KNN > do
9:       if verify(p, k, q) then
10:         R ← R ∪ p
11:      end if
12:   end for
13:   if |KNN| < k then
14:      for all ¡b ∈ BN¿ do
15:         if v.cid is visited first time then
16:            CP ← findPOIinCell(v.cid)
17:            for all p ∈ CP do
18:               if verify(p, k, q) then
19:                  R ← R ∪ p
20:               end if
21:            end for
22:         end if
23:         PQ.enQueue(< d_N(q, b), b, p, v.cid >)
24:      end for
25:   end if
26: end while
27: return R {Rk NN of q}
28: endfunction
```

Line 13 of Algorithm 9 checks whether the number of elements in KNN is less than k, i.e., the number of POIs existing in the area whose distance from $v.n$ is less than k. If so, node $v.n$ is expanded by the procedure ExpandCell and the search is continued. Otherwise, no more RkNNs exist on the path through $v.n$; therefore, node expansion at $v.n$ is not executed.

5.3.3 Simple Path Finder Algorithm Using in RkNN Query

The two procedures, rangeNN and verify, need to determine road network distance between terminal point pairs, s and d, when they are implemented by the IER framework. This search can be realized by referencing the IBDT and BBDT. Fig. 5.4 to Fig. 5.6 show the process flow of the shortest path finder (SPF)[105]. The SPF is controlled by best-first search using priority queue PQ, which in turn manages records constructed as

$$< p, Cost, dfs, fSG, phase >$$

where p is the currently noticed point (i.e., s, d, or a border node), $Cost$ is the lower-bound road network distance between s and d (and is the key used for ordering PQ), dfs (distance-from-source) is the shortest road network distance between s and the currently noticed node p, fSG is the subgraph ID in which p belongs. Here, we use two values of $phase$ to show the progression of the algorithm, PHASE1 (searching state) and PHASE2 (final state).

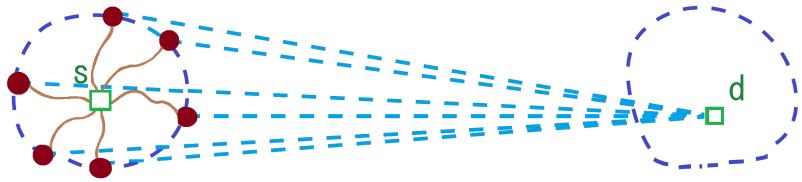


Figure 5.4: Initial state

First, subgraph SG_s, which contains the road segment under s, is determined. Next, $Cost$ is calculated by the equation $Cost = d_N(s, b_i) + d_E(b_i, d)$ for all border

nodes $b_i \in BV_s$ of SG_s. Here $d_E(x, y)$ denotes the Euclidean distance between x and y and $d_N(x, y)$ denotes the road network distance between x and y. In the above equation, $d_N(s, b_i)$ can be obtained by referencing the IBDT of SG_s. In this initial stage (see in Fig. 5.4), the following records are composed and added to PQ. In this processing stage, the records have PHASE1 as their phase value; i.e.,

$$< b_i, d_N(s, b_i) + d_E(b_i, d), 0, SG_s, PHASE1 > \quad for\ all\ b_i \in BV_s$$

If s and d are very close and included in the same subgraph, the distance cannot be obtained via the IBDT. In this case, the distance calculation method is described in Sect. 5.3.5.

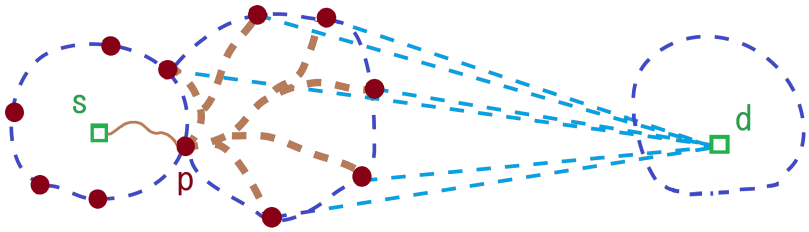


Figure 5.5: Shift to neighboring cell(PHASE1)

As shown in Fig. 5.5, when the *phase* value of the obtained record (e) from PQ is $PHASE1$, the subgraph is shifted to the neighboring subgraph, SG_n. For each subgraph SG_n, $Cost$ is calculated as

$$Cost = e.dfs + d_N(p, b_i) + d_E(b_i, d) \quad (b_i \in BV_n),$$

where BV_n is a border node set of SG_n. The following record is composed and

added to PQ:

$$< b_i, Cost, e.dfs + d_N(p, b_i), SG_n, PHASE1 > .$$

Continuing the process, when a record obtained from PQ reaches a border node of the subgraph containing d (SG_d), the record shown below is constructed and added to PQ.

$$< e.p, e.dfs + d_N(e.p, d), e.dfs, e.fSG, PHASE2 >,$$

where $d_N(e.p, d)$ is obtained from the IBDT of SG_d and $PHASE2$ shows the status that a route between s and d is found.

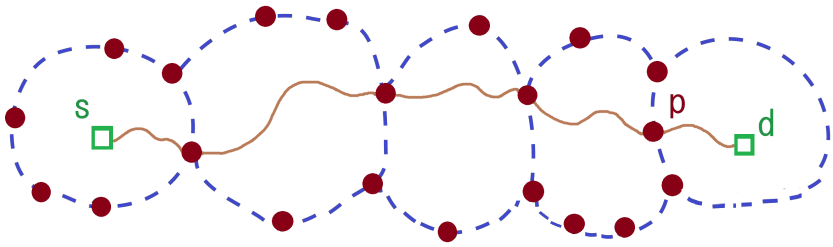


Figure 5.6: Processing states in PHASE2

As described in Fig. 5.6, when the *phase* value of the dequeued record is $PHASE2$, the shortest path distance between s and d has been determined. The fact that the record is dequeued from PQ means it has the minimum $Cost$ value among all records contained in PQ. Therefore, the (shortest) distance is returned and the search process is terminated.

5.3.4 Limitation of referring tables for distance calculation

When s and d are located in the same subgraph, the distance between them cannot be decided via the BBDT or IDBT. However, in this case, the distance can be regarded to be small. In general, a modified A* algorithm, which is hereafter referred to as a pairwise A* (or PWA*) algorithm, can search for the shortest path efficiently when two terminal points are located close to one another.

Fig. 5.7 shows the example in which s and d belong to the same subgraph, SG_a. The road network distance is 9; however, it becomes 13 when calculated locally in SG_a. Therefore, even if s and d belong to the same subgraph, the procedure described in Section 5.3.4 above is necessary to find the global shortest path. In this case, $< d, 9, 0, SG_a, PHASE2 >$ is inserted into PQ, besides the record targeting the border node.

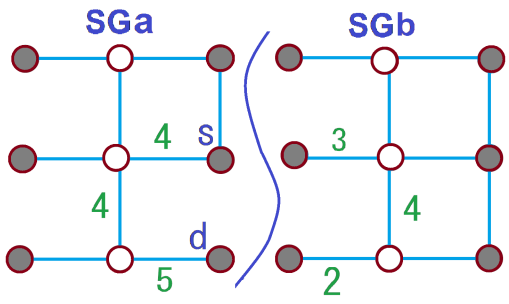


Figure 5.7: Local and global shortest path distance

In the explanation, thus far, s and d are located on network nodes; however, in the usual use, these points are located on the road edges. Generalizing this, when s is located on edge e_s and d is located on edge e_d, the shortest path length can be determined by applying SPF algorithms to the four combinations among both

endpoints of e_s and those of e_d. In this case, the minimum value of distances from s to d via any endpoint of both links is added to the results.

5.4 Experimental results

We evaluated our proposed method by comparing it with the Eager algorithm presented in [73]. Both algorithms are implemented in Java and evaluated on a PC with an Intel Core i7 CPU 960 (3.2GHz) and 9GB memory. Table 5.1 shows the road network maps used in this experiment. In this table, *Adj.List* shows the size of the adjacency list, and BBDT and IBDT are the size of the tables described in previous session.

Table 5.1: Road network maps data used in the experiments

Map name	# of nodes	# of links	Area size	Adj. List	BBDT	IBDT
MapA	16,284	24,914	168 km^2	1.5MB	1.1MB	4.1MB
MapB	109,373	81,233	284 km^2	6.8MB	4.5MB	17.4MB

Table 5.2 Data size used in each map (MB)

Map name	Eager	SMPV	HEPV
Map A	1.5	6.7	30.1
Map B	6.8	28.7	376.1

The adjacency list preparation and the partitioning of the road network into subgraphs were performed by the same processes described in session 4.3. Then the BBDT and IBDT tables were prepared for each subgraph. The average number of

nodes in a subgraph is approximately 240 for these two types of maps. We used the POI data generated by a pseudo-random sequence.

Table 5.2 compares the data size of the adjacency list (used primarily by the Eager and Lazy algorithms), SMPV (total of the adjacency list, BBDT, and IBDT), and HEPV [22]. The data size of SMPV is approximately 4–5 times larger than that of the Eager algorithm; however, SMPV drastically reduces the data size in comparison with usual hierarchical distance materialization method, HEPV.

We generated several POI sets on the road network nodes by pseudorandom sequences, changing density D. For example, $D = 0.01$ indicates that a POI exists once every 100 nodes. The experimental result is shown in Fig. 5.8. The horizontal line shows the varying of POI density and the vertical line shows the processing time in second.

Firstly, we performed experiment to evaluate the shortest path finding in SMPV comparing with pair-wise A* algorithm (PWA*). We searched 10 nearest neighbors (10NN) based on IER strategy (i.e. candidate points are searched in Euclidean distance, and then the shortest path between the query point and each candidate point is searched using A* algorithm and SMPV.

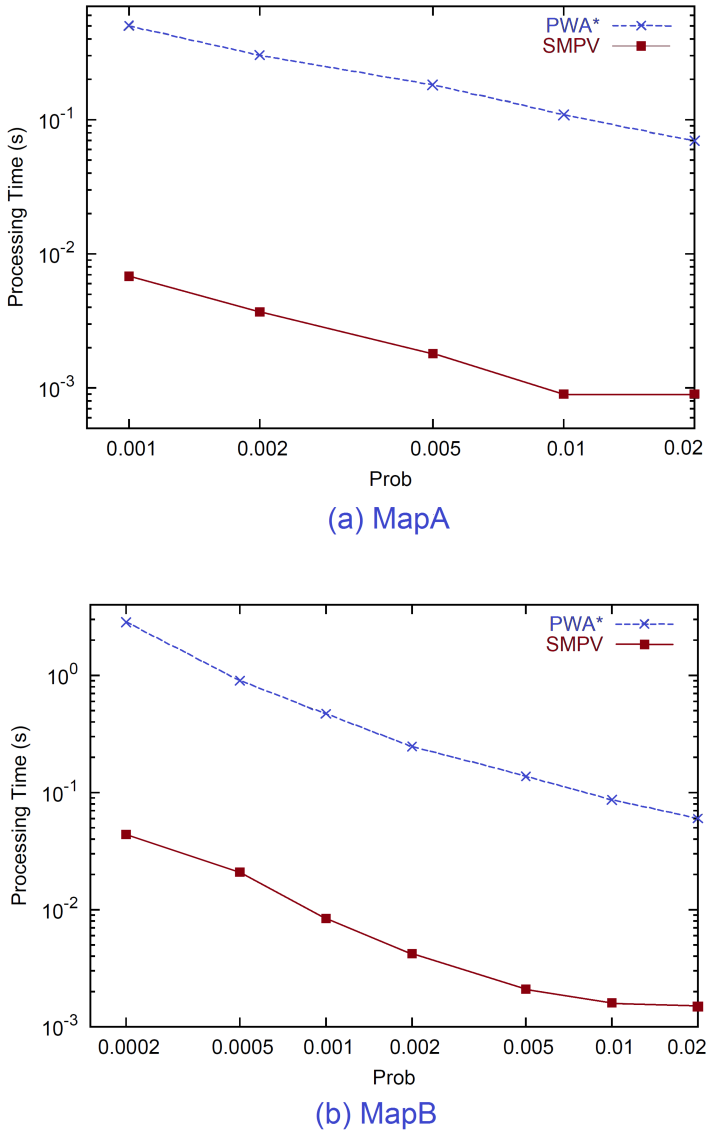


Figure 5.8: Processing time for kNN queries

Fig. 5.8(a) is the result for MapA and Fig. 5.8(b) is for MapB respectively. According to this experimental result, the processing time of SMPV to search kNN is 100 times faster than PWA* algorithm.

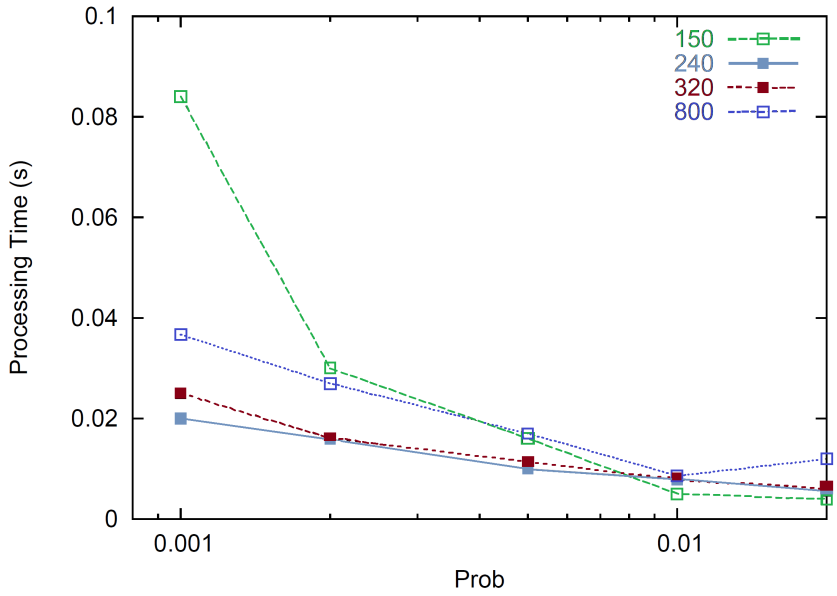


Figure 5.9: Varying average nodes numbers in subgraph

Then we have performed experiment on SMPV by varying the average number of nodes in subgraphs. By comparing the processing time of SMPV, the average nodes number 240 has the minimum processing time when POI is sparsely distributed and the average nodes number 800 has the minimum processing time when POI distribution is denser than 0.01 (see in Fig. 5.9).

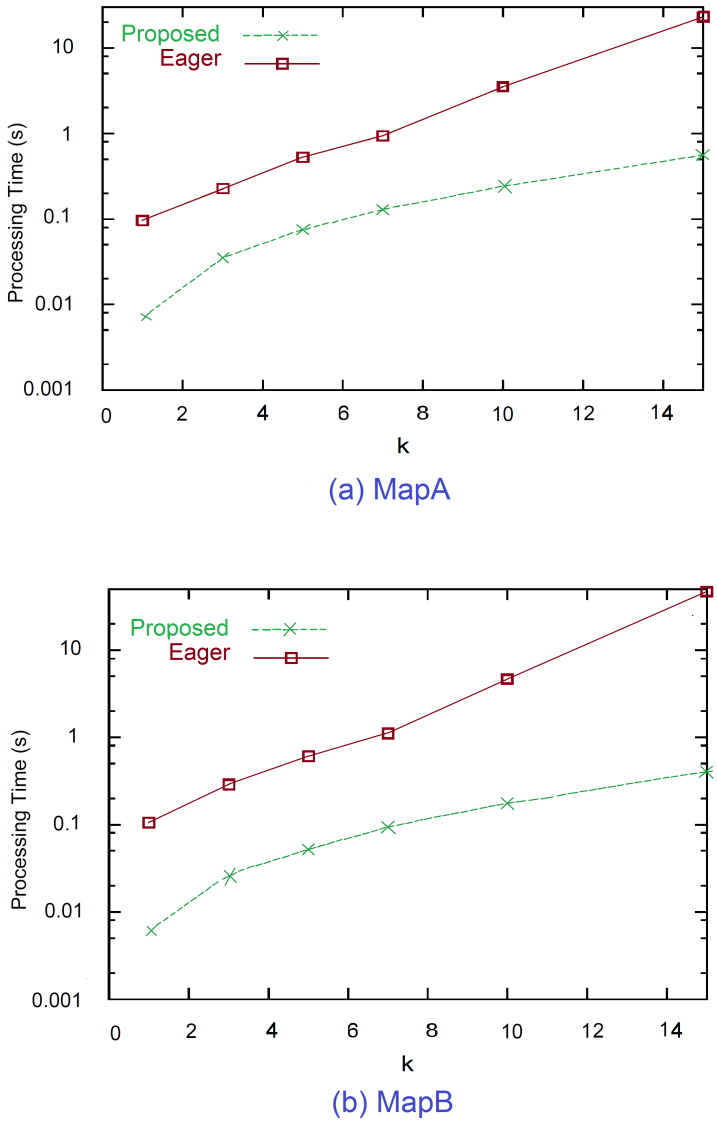
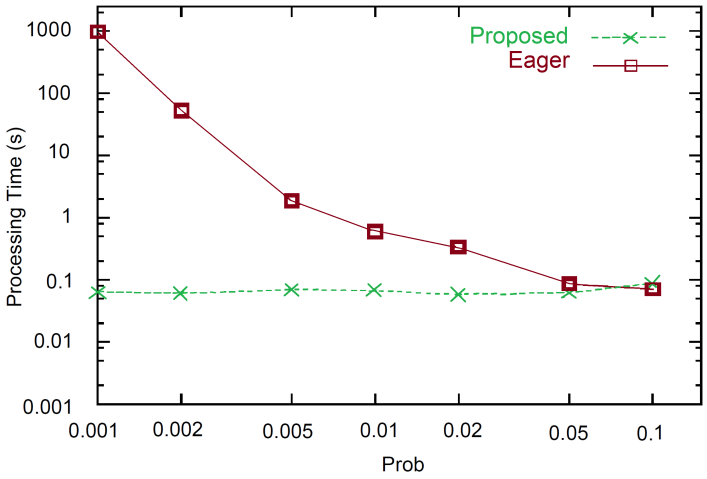


Figure 5.10: Processing time for varying k value

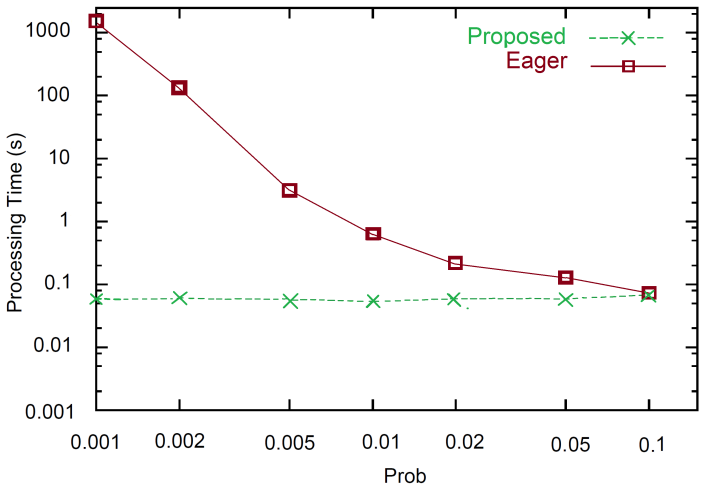
When the POI density is 0.01 and varying k values, the processing times of Eager algorithm and proposed method are described in Fig. 5.10(a) and 5.10(b) for MapA and MapB respectively.

In these figures, the horizontal axes show varying k value and the vertical axes show the processing time in seconds. The tendency of the experimental results is almost the same in both maps. The processing time of the Eager algorithm increases sharply with an increase k because of the expandability of the search area. In contrast, the proposed algorithm increases linearly with increasing k values. In addition, the difference between the processing times of Eager algorithm and our proposed method in MapB becomes larger than that difference in MapA.

Then we execute the experiments with varying POI density (D). Here k value is fixed to 5. Fig 5.11 compares the processing times of Eager algorithm and the proposed method. When the POI density is low, the processing time of Eager algorithm increases sharply. However, the processing time of the proposed algorithm is still low in same condition. When the POI density is high, Eager algorithm performs well because the size of the search area decreases with increase in POI density. According to the results in both maps, our proposed algorithm shows stable characteristics independent of the probability which is essential for query processing in LBS.



(a) MapA



(b) MapB

Figure 5.11: Processing time for varying Prob value

5.5 Summary

We propose an RkNN query algorithm using the simple distance materialization approach suitable for LBS. Comparing with conventional hierarchical network distance materialized method, HEPV, the amount of data used in proposed methods decreases. Referring the experimental results, the proposed method outperforms the existing method (Eager algorithm) for finding RkNNs on real road network especially when the data points are sparsely distributed or k value is large. In addition, the proposed method has stable characteristic and low processing time even the Eager algorithm increases the processing time in low POI density.

CHAPTER 6

Conclusion

We conclude our studies in this thesis by the following topics

1. We proposed a realtime monitoring on Moving Objects by using frequently route routes for both thick client and thin client
2. We proposed an efficient shortest path finder algorithm based on materialized path view method
3. We applied our SPF algorithm with materialized path view to Reverse kNN query which is essential in LBS

For (1) we proposed the real time monitoring of moving objects based on road network distances. In recent time, studies on a real time monitoring of moving objects (cars and humans) becomes an active role for some location related applications such as intelligent transportation system, security system for children, wild animal monitoring etc. In realtime monitorings, the main issue is how to get high accuracy tracking of moving objects with less communication between moving objects and server. Some studies adopted to achieve this goal by sharing routes information

between moving objects and sever but it is not in road network or moving object is not in dynamic routes. In this thesis, we proposed a real time monitoring method aiming for both thick client, such as car navigation system, and thin client, such as mobile phone user. Using the " frequently used route " information, we offered high scalability monitoring system with empirical comparisons of the proposed method and the conventional dead-reckoning on road network method. And also we applied an efficient map-matching algorithm to track the moving object's movements. According to the experiment results, our proposed method outperformed the conventional method DRRN by factors from 2 to 4 in thick client and 4 to 6 in thin client.

For (2), we proposed an efficient shortest path finding algorithm using distance materialization method which is very useful for LBS application. Due to the sizes of the real road network maps, applied this method in three variations SPFLM, SPFMM and SPFFM. According to the experimental results, our proposed method reduced the used data amount comparing with the existing hierarchical network distance materialized methods; HEPV and HiTi. Moreover, not only SPFLM reduced the data amount drastically, SPFMM and SPFFM also achieved similar time efficiency with the hierarchical encoded path view.

Consequently, when the distance between two points was large, SPFLM outperformed PWA* substantially, nevertheless the SPFLM used a small amount of pre-computation data. LBS is apt to request for the shortest path searches over rather than nearly located points, and the operation is repeated over a large number of times in a query; for example as in the incremental Euclidean restriction strategy. In this situation, the relative search speed of PWA* increases, because the hit ratio in LRU buffer managing adjacency list is increased. kNN queries evaluated in this paper is for such example. When the density of POI was high, the difference of the processing times between SPFLM and PWA* became small. On the other hand, SPFMM and SPFFM outperformed the other methods even in such situation.

Regarding (3), we applied our distance materialization method to Reverse kNN query. For RkNN query, expanding the search area of the proposed method in concentric circles is similar to that in Eager algorithm. The algorithm finds data points in the range at every border node (i.e. the radius is same with the distance between the query point and the border node) which centered at the query point. If data points are found within the range, the results are checked for whether they are truly RkNN of the query point using verification step. In most existing methods for RkNN query, *range*NN searches are performed at every network node and that causes a long processing time. In our proposed method, the *range*NN procedure was performed only on border nodes. This limitation drastically reduced the overall total time of invoking rangeNN. In addition, IER adaptation to the rangeNN and verify procedures contributed to reduce the overall processing time. Consequently, the proposed method performed RkNN on a road network significantly well, especially when the density of POIs was low or k value was large. Moreover, our proposed method showed stable characteristic independent of the POI density.

We experimentally evaluated all proposed algorithms to prove their accuracy and efficiency as well. For the future study, if the processing time in distance materialization method can be advanced applying in some queries, we will attempt to adopt this method for some spatial queries in real road network.

Acknowledgement

First of all, I would like to thank Saitama University for giving me a chance to fulfill my dream of being a doctoral student. I would also like to express my sincere gratitude to my supervisor Professor Yutaka Ohsawa who has the attitude and the substance of a genius. I could not thank enough for all the kindness he extended during my study. Without his guidance and persistent help, this dissertation would not have been possible.

I would like to acknowledge the support of Professor Masao Sakauchi and Professor Noboru Sonehara from National Institute of Informatics for my Ph.D study. I am also grateful to the Co-supervisors and the members of examiners, Professor emeritus Hitoshi Maekawa, Professor Tetsuya Shimamuya, Professor Toshinori Yamada and Professor Yoshinori Kobayashi who encouraged me to achieve a success in all presentations and spent extra time to discuss, comment and advice me in the preparation of my thesis.

In addition, I would like to deepest thank to Heiwa Nakajima Scholarship foundation who had provided the financial support to undertake the fulltime study at university. I am thankful to my dear seniors, friends and all members of Ohsawa department for their kind help to all challenges.

Further, I dedicate to my husband and my family members for their support and help in all hours and days that this thesis consumed. I would like to express my utmost gratitude to my parents. I am forever grateful for the depth of their love,

and the warm and gentle care they provided.

Finally, I would like to heartfelt thank to my beloved mother who wanted me to finish this long journey. If I had a chance to meet her for one last time, I would just tell her that I fulfill her wish, hug her tightly, rest my head on her shoulder and cry till my tears run dry.

References

[1] Papadias, D., Zhang, J., Mamoulis, N., Tao, Y.: Query processing in spatial network databases. In: Proc. 29th VLDB. (2003) 790–801

[2] Guttman, A.: R-Trees: a dynamic index structure for spatial searching. In: Proc. ACM SIGMOD Conference on Management of Data. (1984) 47–57

[3] Beeri, C., Kanza, Y., Safra, E., Sagiv, Y.: Object fusion in geographic information systems. In: Proc. 13th VLDB. (2004) 816–827

[4] George, B., Kim, S., Shekhar, S.: Spatio-temporal network databases and routing algorithms: a summary of results. In: Proc. 10th SSTD. (2007) 460–477

[5] Samet, H.: Issues, developments, and challenges in spatial databases and geographic information systems (gis). In: Proc. 9th VLDB. (2001) 1

[6] Shekhar, S., Coyle, M., Goyal, B., Liu, D., Sarkar, S.: Data models in geographic information systems. In: Commun ACM40(4). (1997) 103–111

[7] Wang, H., Zimmermann, R.: Location-based query processing on moving objects in road networks. In: Proc. VLDB. (2007) 23–28

[8] Dijkstra, E.W.: A note on two problems in connection with graphs. Numeriche Mathematik **1** (1959) 269–271

[9] Shekhar, S., Chawla, S.: Spatial databases. In: A Tour. Prentice Hall, Englewood Cliffs. (2002)

[10] Chen, M., Chowdhury, R., Ramachandran, V.: Priority queues and dijkstra's algorithm. In: Technical report, UTCS Technical Report. (2007) 07–54

[11] Johnson, D.B.: Efficient algorithms for shortest paths in sparse networks. Journal of the ACM **24**(1) (1977) 1–13

[12] L.Fu, D.Sun, L.R.Rilett: Heuristic shortest path algorithms for transportation applications: State of the art. Computers and Operations Research **33** (2006) 3324–3343

[13] Hart, P.E., Nilsson, N.J., Raphael, B.: A formal basis for the heuristic determination of minimum cost paths. IEEE Transactions of Systems Science and Cybernetics **ssc-4**(2) (1968) 100–107

[14] Jing, N., Huang, Y.W., Rundensteiner, E.A.: Hierarchical optimization of optimal path finding for transportation applications. In: Proc. Fifth Int'l Conf. Information and Knowledge Management. (1996) 268–276

[15] Samet, H., Sankaranarayanan, J., Alborzi, H.: Scalable network distance browsing in spatial databases. In: Proc. of the ACM SIGMOD conference. (2008) 43–54

[16] Shekhar, S., Fetterer, A., Goyal, B.: Meterialization trade-offs in hierarchical shortest path algorithms. In: 5th International Symposium on Large Spatial Databases. (1997) 94–111

[17] Krozel, J., II, D.A.: Intelligent e-optimal path prediction for vehicular travel. In: Proc. IEEE Trans. (1995) 345–353

[18] Shekhar, S., Kohli, A., Coyle, M.: Path computation algorithms for advanced traveler informa. In: Proc. 9th Data Eng. (1993) 31–39

[19] Agrawal, R., Jagadish, H.: Materialization and incremental update of path information. In: Proc. 5th Data Eng. (1989) 374–383

[20] Huang, Y., Jing, N., Rundensteiner, E.: Semi-materialized view approach for route maintenance in intelligent vehicle highway systems. In: Proc. 2nd ACM Workshop. (1994) 144–151

[21] Jung, S., Pramanik, S.: HiTi graph model of topographical road maps in navigation systems. In: Proceedings of the 12th International Conference on Data Engineering. (1996) 76–84

[22] Jung, S., Pramanik, S.: An efficient path computation model for hierarchically structured topographical road maps. IEEE Transactions on Knowledge and Data Engineering **14**(5) (2002) 1029–1046

[23] Bar-Noy, A., Kessler, I., Sidi, M.: Mobile users: To update or not to update? Wireless Networks **1**(2) (1995) 175–185

[24] Pitoura, E., Samaras, G.: Locating objects in mobile computing. IEEE Trans. on Knowledge and Data Engineering **13**(4) (2001) 571–592

[25] Wolfson, O., Sistla, A.P., Chamberlain, S., Yesha, Y.: Updating and queying databases that track mobile units. Distributed and Parallel Databases **7**(3) (1999) 257–287

[26] Zhou, J., Leong, H.V., Lu, Q., Lee, K.C.: Generic adaptive moving object tracking algorithms. In: International Conference on Parallel Processing. (2006) 93–100

[27] Wolfson, O., Chamberlain, S., Dao, S., Jiang, L., Mendez, G.: Cost and imprecision in modeling the position of moving objects. In: Proc. 14th ICDE. (1998) 588–596

[28] Wolfson, O., Xu, B., Chamberlain, S., Jiang, L.: Moving objects databases: Issues and solutions. In: Proc 10th SSDBM. (1998) 111–122

[29] Tiesyte, D., Jensen, C.S.: Efficient cost-based tracking of scheduled vehicle journeys. In: The 9th MDM. (2008) 9–16

[30] Tiesyte, D., Jensen, C.S.: Similarity-based prediction of travel times for vehicles traveling on known routes. In: Proc. ACM GIS'08. (2008)

[31] Ding, Z., Güting, R.H.: Managing moving objects on dynamic transportation networks. In: Proc. 16th SSDBM. (2004) 287–296

[32] Čivilis, A., Jensen, C.S., Nenortaitė, J., Pakalnis, S.: Efficient tracking of moving objects with precision guarantees. Technical Report TR-5, Department of Computer Science, Aalborg University (2004)

[33] Čivilis, A., Jensen, C.S., Pakalnis, S.: Techniques for efficient road-network-based tracking of moving objects. IEEE Trans. on Knowledge and Data Engineering **17**(5) (2005) 698–712

[34] Frentzos, E.: Indexing objects moving on fixed networks. In: Proc. 8th SSTD. (2003) 289–305

[35] Ku, W.S., Zimmermann, R., Wang, H., Wan, C.N.: Adaptive nearest neighbor queries in travel time network. In: Proc. ACM GIS'05. (2005) 210–219

[36] Mouratidis, K., Yiu, M.L., Papadias, D., Mamoulis, N.: Continuous nearest neighbor monitoring in road networks. In: Proc. 32th VLDB. (2006) 43–54

[37] Hsueh, Y.L., Zimmermann, R., Wang, H., Ku, W.S.: Partition-based lazy updates for continuous queries over moving objects. In: Proc ACM GIS '07. (2007)

[38] Brakatsoulas, S., Pfoser, D., Salas, R., Wenk, C.: On map-matching vehicle tracking data. In: Proc. 31st VLDB Conference. (2005) 853–864

[39] Greenfeld, J.S.: Matching GPS observations to locations on a digital map. In: Proc. 81th Annual Meeting of the Transportation Reseach Board. (2002)

[40] Quddus, M.A., Ochieng, W.Y., Zhao, L., Noland, R.B.: A general map matching algorithm for transport telematics applications. GPS Solutions **7** (2003) 157–167

[41] Yin, H., Wolfson, O.: A weight-based map matching method in moving objects databases. In: Proc. 16th SSDBM. (2004) 437–438

[42] Roussopoulos, N., Kelly, S., Vincent, F.: Nearest neighbour queries. In: Proceedings ACM SIGMOD Conference on Management of Data. (1995) 71–79

[43] Hjaltason, G.R., Samet, H.: Distance browsing in spatial databases. ACM Transactions on Database Systems **24**(2) (1999) 265–318

[44] Berchtold, S., Keim, D., Seid, H.K.T.: Indexing the solution space: A new technique for nearest neighbor search in high-dimensional space. In: TKDE. (2000) 45–57

[45] Jensen, C., Kolarvr, J., Pedersen, T., Timko, I.: Nearest neighbor queries in road networks. In: Proc.11th ACM International Symposium on Advances in Geographic Information Systems (GIS'03). (2003) 1–8

[46] Hu, H., Lee, D.L., Xu, J.: Fase nearest neighbor search on road networks. In: Proceeding of the 10th International Conference on Extending Database Technology. (2006) 186–203

[47] Kolahdouzan, M., Shahabi, C.: Voronoi-based K nearest neighbor search for spatial network databases. In: Proc. 30th VLDB. (2004) 840–851

[48] Shang, S., Deng, K., Xie, K.: Best point detour query in road networks. In: Proc. ACM GIS. (2010)

[49] Berchtold, C.S., Keim, D., Kriegel, F.K.H.: On optimizing nearest neighbor queries in high-dimensional data spaces. In: Proc.ICDT 2001. (2001) 435–449

[50] Uemiya, T., Matsumoto, Y., Koizumi, D., Shishibori, M., Kita, K.: Fast multidimensional nearest neighbor search algorithm based on ellipsoid distance. In: International Journal of Advanced Intelligence Volume 1, Number 1. (2009) 89–107

[51] Tao, Y., Yi, K., Sheng, C., Kalnis, P.: Quality and efficiency in high dimensional nearest neighbor search. In: SIGMOD. (2009)

[52] Ma, X., Shekhar, S., Xiong, H., Zhang, P.: Exploiting a page-level upper bound for multi-type nearest neighbor queries. In: ACM GIS'06. (2006) 179–186

[53] Wu, S., Chuang, K., Chen, C., Chen, M.: An itinerary-based knn query processing algorithm for mobile sensor networks. In: ICDE. (2007) pp.456–465

[54] Kang, J., Mokbel, M., Shekhar, S., Xia, T., Zhang, D.: Continuus evaluation of monochromatic and bichromatic reverse nearest neighbors. In: Proc. ICDE'07. (2007) 806–815

[55] Gao, Y., Chen, G., Li, Q., Zheng, B., Li, C.: Processing mutual nearest neighbor queries for moving object trajectories. In: IEEE 9th MDM. (2008) 116–123

[56] B.Tao, Kumar, F.: Visible reverse k-nearest neighbor queries. In: Proc. ICDE09. (2009) 1203–1206

[57] Berchtold, S., Ertl, B., Keim, D., Kriegel, H., Seidl, T.: Fast nearest neighbor search in high-dimensional space. In: Proc. 14th Conference on Data Engineering. (1998) 23–27

[58] Cheung, K.L., chee Fu, A.W.: Enhanced nearest neighbour search on the R-tree. SIGMOD Record **27**(3) (1998) 16–21

[59] Lang, C., Singh, A.: A framework for accelerating high-dimensional *nn*-queries. In: Technical report, TRCS01.04, Department of Computer Science, Uni- versity of California, Santa Barbara. (2001)

[60] Ferhatosmanoglu, H., Stanoi, I., Agrawal, D., Abbadi, A.: Constrained nearest neighbor queries. In: Proc. SSTD,LNCS 2121. (2001) 257–276

[61] Hjaltason, G.R., Samet, H.: Incremental distance join algorithms for spatial databases. In: Proceedings ACM SIGMOD Conference on Management of Data. (1998) 237–248

[62] Athitsos, V., Potamias, M., Papapetrou, P., Kollios, G.: Nearest neighbor retrieval using distance-based hashing. In: Proc. ICDE08. (2008) 327–336

[63] Zhang, Z., Yang, Y., Tung, A., Papadias, D.: Continuous k-means monitoring over moving objects. In: Proc. IEEE Trans. K and D Eng., Vol.20, No.9. (2008) 1205–1216

[64] Gao, Y., Z.Baihua: Continuous obstructed nearest neighbor queries in spatial databases. In: Proc.SIGMOD. (2009) 577–589

[65] Kamel, I., Faloutsos, C.: On packing r-trees. In: Proc. 2nd ACM Intl. Conf. on Information and Knowledge Management. (1993) 490–499

[66] Faloutsos, C., Kamel, I.: Beyond uniformity and independence: Analysis of r-trees using the concept of fractal dimension. In: Proc. 13th ACMSIGACT-SIGMODSIGART Symposium on Principles of Database Systems. (1994) 4–13

[67] Theodoridis, Y., Sellis, T.: A model for the prediction of r-tree performance. In: Proc. 15th ACM SIGACT-SIGMODSIGART Symposium on Principles of Database Systems. (1996) 161–171

[68] Theodoridis, Y., Stefanakis, E., Sellis, T.: An efficient cost model for spatial queries using r-trees. In: Technical report, KDBSLAB-TR-97-01, Department

of Electrical and Computer Engineering, Computer Science Division, National TechnicalUniversity of Athens. (1997)

[69] Proietti, G., Faloutsos, C.: I/o complexity for range queries on region data stored using an r-tree. In: Proc. 15th International Conference on Data Engineering. (1999) 628–635

[70] Korn, F., Muthukrishnan, S.: Infuluence sets based on reverse nearest neighbor queries. In: ACM SIGMOD Record. Volume 29. (2000) 201–212

[71] Stanoi, I., Agawal, D., Abbadi, A.E.: Reverse nearest neighbor queries for dynamic databases. In: Proc. of 2000 ACM SIGMOD Workshop on Reserch Issues in Data Mining and Knowledge Discovery. (2000) 44–53

[72] Tao, Y., Papadias, D., Lian, X.: Reverse $k$nn search in arbitrary dimensionality. In: Proceedings of the 30th VLDB Conference. (2004) 744–755

[73] Yiu, M.L., Papadias, D., Mamoulis, N., Tao, Y.: Reverse nearest neighbor in large graphs. IEEE Transaction on Knowledge and Data Engineering **18**(4) (2006) 1–14

[74] Safar, M., Ibrahimi, D., Taniar, D.: Voronoi-based reverse nearest neighbor query processing on spatial netowrks. Multimedia Systems **15** (2009) 295–308

[75] Cheema, M.A., Zhang, W., Lin, X., Zhang, Y., Li, X.: Continuous reserse k nearest neighbors queries in euclidean space and in spatial netowrks. VLDB Journal **21** (2012) 69–95

[76] Stojanovic, D., Paradopoulos, A., Predic, B., Kajan, S., Nanopoulos, A.: Continuous range monitoring of mobile objects in road networks. In: Data and Knowledge Engineering. (2008) 77–100

[77] Hu, H., Lee, D.: Range nearest-neighbor query. In: IEEE Trans. Knowledge and Data Engineering,Vol18. (2006) 78–91

[78] Shekhar, S., Yoo, J.S.: Processing in-route nearest neighbor queries: A comparison of alternative approaches. In: Proc. ACM GIS '03. (2003) 9–16

[79] Gao, Y., Chen, G., Li, Q., Zheng, B., Li, C.: Processing mutual nearest neighbor queries for moving object trajectories. In: The Ninth International Conference on Mobile Data Management. (2008) 116–123

[80] Chen, Z., Shen, H.T., Zhou, X., Yu, J.X.: Monitoring path nearest neighbor in road networks. In: SIGMOD'09. (2009) 591–602

[81] Jin, C., Guo, W.: Efficiently monitoring nearest neighbors to a moving object. In: ADMA 2007. (2007) 239–251

[82] Kolahdouzan, M.R., Shahabi, C.: Continuous K nearest neighbor queries in spatial network databases. In: Proceedings of the Second Workshop on Spatio-Temporal Database Management. (2004) 33–40

[83] Song, M., Park, K., Im, S., Kong, K.S.: Nearest neighbor queries for R-Trees: Why not bottom-up? In: 8th International Conference on Database Systems for Advanced Applications (DASFF 2006). (2006) 910–919

[84] Chen, L., Lian, X.: Dynamic skyline queries in metric spaces. In: Proc. 11th International conference on Extending database technology. (2008) 333–343

[85] Deng, K., Zhou, X., Shen, H.T.: Multi-source skyline query processing in road networks. In: Proceeding of IEEE 23rd International Conference on Data Engineering. (2007)

[86] M.Sharifzadeh, C.Shahabi:: The spatial skyline queries. In: Proc. VLDB. (2006) 751–762

[87] Song, Z., Roussopoulos, N.: K-nearest neighbour search for moving query point. In: SSTD 2001. (2001) pp.79–96

[88] Nutanong, S., Zhang, R., Tanin, E., Kulik, L.: The V*-diagram: A query-dependent approach to moving KNN queries. In: Proc. PVLDB'08. (2008) 1095–1106

[89] Kolbe, D., Zhu, Q., Pramanik, S.: On k-nearest neighbor searching in non-ordered discrete data spaces. In: ICDE. (2007) 426–435

[90] Comer, D.: The ubiquitous b-tree. In: ACM Computing Surveys, Vol 11, No.2. (1979) 121–137

[91] Bertino, E., Catania, B., Chiesa, L.: Definition and analysis of index organizations for object-oriented database systems. In: Proc. International Workshop on Advances in Databases and Information Systems. (1998) 65–108

[92] de Almeida, V.T., Güting, R.H.: Indexing the trajectories of moving objects in networks. Technical report, Fernuniversität Hagen (2004)

[93] Wang, L., Y.Zheng, Ma, X.W.: A flexible spatio-temporal indexing schema for large-scale gps track retrieval. In: Proc. IEEE 9th MDM. (2008) 1–8

[94] Ding, Z.: Utr-tree:an index structure for the full uncertain trajectories of network-constrained moving objects. In: Proc.IEEE 9th. (2008) 33–40

[95] Saltenis, S., Jensen, C.: Indexing of moving objects for location-based services. In: Proc. International Conference on Data Engineering (ICDE). (2002) 463–472

[96] Theodoridis, Y., Vazirgannis, M., Sellis, T.: Spatio-temporal indexing for large multimedia applications. In: Proceedings of the 3rd IEEE International Conference on Multimedia Computing and Systems (ICMCS). (1996) 441–448

[97] Nascimento, M., Silva, J.: Towards historical r-trees. In: Proc. 13th ACM. (1998)

[98] Pfoser, D., Jensen, C., Theodoridis, Y.: Novel approaches to the indexing of moving object trajectories. In: Proc. 26th International Conference on Very Large Databases. (2000)

[99] Jensen, C.S., Pakalnis, S.: TRAX – real-world tracking of moving objects. In: Proceeding of 33rd VLDB. (2007) 1362–1365

[100] Liu, X., Karimi, H.A.: Location awareness through trajectory prediction. Computers, Environment and Urban Systems **30** (2006) 741–756

[101] Karimi, H.A., Liu, X.: A predictive location model for location-based services. In: Proc. ACM GIS'03. (2003) 126–133

[102] Liu, B., Tay, J.: Using knowledge about the road network for route finding. In: Proceedings of the 11th Conference on Artificial Intelligence for Applications. (1995) 306 –312

[103] Htoo, H., Ohsawa, Y., Sonehara, N., Sakauchi, M.: Aggregate nearest neighbor search methods using SSMTA* algorithm. In: 16th ADBIS, LNCS 7503. (2012) 181–194

[104] Liu, X., Schrack, G.: Encoding and decoding the Hilbert order. Software – Practive and Experience **26**(12) (1996) 1335–1346

[105] Hlaing, A.T., Htoo, H., Ohsawa, Y., Sonehara, N., Sakauchi, M.: Shortest path finder with light materialized path view for location based services. In: Proc. WAIM2013. Volume LNCS7923. (2013) 229–234

Author's Publications

[1] Yutaka Ohsawa, Kazuhisa Fujino, Htoo Htoo, Aye Thida Hlaing, Noboru Sonehara. Real-time Monitoring of Moving Objects Using Frequently Used Routes, DASFAA2011, LNCS 6588, pp.119-133 (2011)

[2] Aye Thida Hlaing, Yutaka Ohsawa, Htoo Htoo, Noboru Sonehara, Masao Sakauchi. Shortest path finder with light materialized path view for location based services, WAIM2013, LNCS7923, pp. 229-234 (2013)

[3] Aye Thida Hlaing, Yutaka Ohsawa, Htoo Htoo. Efficient Reverse kNN Query Algorithm on Road Network Distances Using Partitioned Subgraph, SeCoGIS2014, LNCS 8823, pp.212-217 (2014)

[4] 橋本 知宜・Aye Thida Hlaing・藤野 和久・大沢 裕：道路網上での距離に基づく k-NN 経路探索, 曽根原: 登 第１９回地理情報システム学会 学術研究発表大会，2D2，(2010.10)

[5] 茂木恭兵, Aye Thida Hlaing, 大沢裕, 経路履歴を用いた車両実時間モニタリング, 曽根原登: 第 20 回地理情報システム学会学術研究発表大会, F-5-1,(2011.10)

[6] Yutaka Ohsawa, Aye Thida Hlaing, Light Materialized Path View for Location Based Services, ICCA2014, pp.210-216, 2014