Light Materialized Path View for Location Based Services

Yutaka Ohsawa and Aye Thida Hlaing
Saitama University
ohsawa@mail.saitama-u.ac.jp

Abstract

This paper proposes 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. Through the results, we show the proposed algorithm outperforms the conventional methods.*

1. Introduction

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 (*k*NN 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 [1]. 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.

This paper proposes 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 materia-lization methods.

The rest of the paper is organized as follows. Section 2 summarizes related work. Section 3 proposes the shortest path finder algorithm. The
proposed methods and the conventional methods are evaluated experimentally in Section 4. Section 5 summarizes the present paper.

2. Related work

A shortest path query is a basic operation in several types of queries based on the road network distance, for example, *k*NN queries, ANN queries, CNN queries, and trip planning queries. Shortest path query algorithms have been studied since 1950's and several data structures and algorithms [2] 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 and A* algorithm 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. Jing et al. [3] 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.

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. [3] 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 SGi. 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.

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 [4]. This method also materializes distance between two nodes in the 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. [5] 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. This point is the main difference with the discussion developed in the rest of the paper.

3.Shortest Path Finder

3.1.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 *SGi* in the rest of the paper where there is no ambiguity.

Fig. 1(a) shows an example of a road network graph, here small circles are nodes and lines are edges. Fig. 1 (b) depicts a partition of the graph shown Fig. 1(a). In this partition, the nodes shown by black dots 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 *SGi* is denoted by *BVi*. 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.

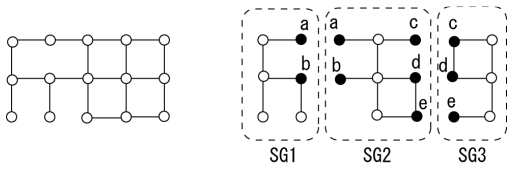
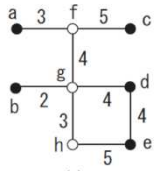


Figure1. Flat graph and its partition



	a	b	c	d	e
f	3	6	5	8	12
g	7	2	9	4	8
h	10	5	12	7	5

(c)

	a	b	c	d	e
a	0	9	8	11	15
b	9	0	11	6	10
c	8	11	0	13	17
d	11	6	13	0	4
e	15	10	17	4	0

(b)

	a	b	c	d	e	f	g	h
a	0	9	8	11	15	3	7	10
b	9	0	11	6	10	6	2	5
c	8	11	0	13	17	5	9	12
d	11	6	13	0	4	8	4	7
e	15	10	17	4	0	12	8	5
f	3	5	6	8	12	0	4	7
g	7	2	9	4	8	4	0	3
h	10	5	13	7	5	7	3	0

(d)

Figure 2. Distance Tables

Fig. 2(a) extracts SG2 from Fig. 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 the rest of the paper, we assume the weight as the length of the link. Fig. 2(b) shows the shortest path length between every two border nodes in SG2. 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. 2(c) 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 matrices of Fig. 2(c) is also necessary to obtain the distance between a border node and the destination point.

Fig. 2(d) shows the node-to-node distance table (NNDT), listed distances of all combinations of the nodes in SG2. This table is used to acquire the distance between two arbitrarily specified nodes. Either IBDT or NNDT is used alternatively in the SPF algorithms described in Sect. 3.3.

3.2.Simple Path Finder Algorithm

Fig. 3 shows the processing flow of the shortest path finder (SPF). 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*. *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, *SGs*, 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 $b_i \in$ BVs of *SGs*. 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.

$<b_i, d_E(s,b_i)+d_E(b_i,d), 0, SGs, PHASE0> \forall b_i \in BVn$

Next, a record (*e*) that has minimum *Cost* value is dequeued from the PQ as shown in Fig. 3(b). At the beginning of the processing, *e.phase* is PHASE0. 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. 3.3. *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.

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

When the *phase* value of the obtained record (*e*) from the PQ is PHASE1 (see Fig. 3 (c)), 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 *SGn*, *Cost* is calculated by the following equation.

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

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

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

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.

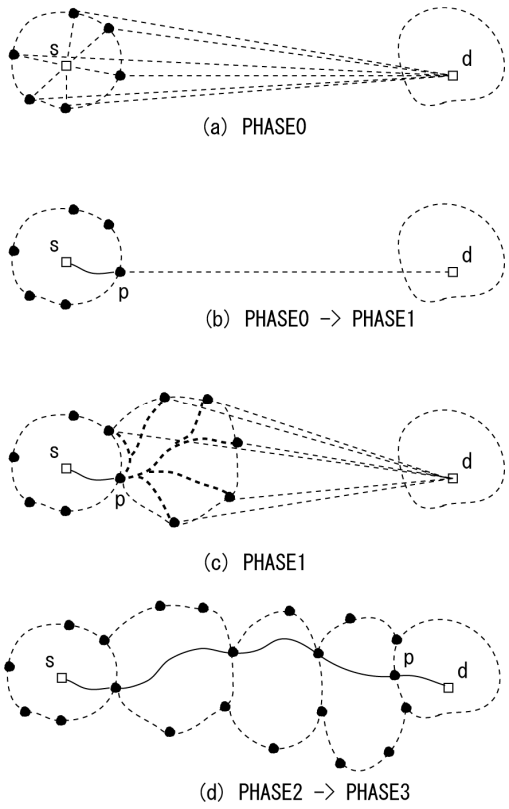


Figure 3. Processing flow of SPF

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.

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

The way how to determine $d_N(e.p,d)$ is described in Sect. 3.3.

When the *phase* value of the dequeued record is PHASE3, the shortest path distance between *s* and *d* has been determined. 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 and returned, and then the searching process is terminated.

3.3.Distance calculation method inside subgraph

This section describes three variations of SPF in Sect. 3.2.; 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.

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 the algorithm described in Sec. 3.2 is also necessary even when *s* and *d* are located in the same subgraph.

Table 1 shows the tables described in Sect. 3.1 used in the three SPF algorithms.

Table 1. Used tables in each SPF algorithm

Data table	SPFLM	SPFMM	SPFFM
BBDT	✓	✓	✓
IBDT		✓	
NNDT			✓
Adj.List	✓	✓	

4. 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 2 shows the road network maps used in this experiment.

Table 2. Road network maps used in the experiments

Map name	No. nodes	No. links	Adj. list size
MapS	16,284	24,914	1.5M
MapM	109,373	81,233	6.8MB
MapL	465,245	638,282	39.7MB

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.

Table 3. Data size (MB)

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

Fig.4 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. 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. 5(a) shows the average processing time spent to determine 10 shortest paths on MapS; Fig. 5(b) and (c) show the results of the same experiments over MapM and MapL, respectively.

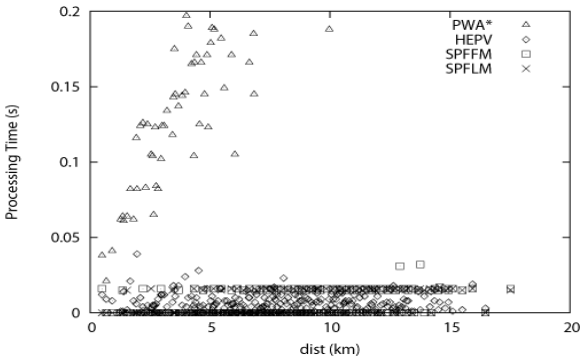


Figure 4. Processing time when s and d are places on nodes (MapS)

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.

Fig. 5(d) 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.

5. Conclusion

This paper proposes a shortest path search algorithm and its three variations using the light distance materialization that are suitable for LBS. The data amount of the presented 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 HEPV.

Consequently, when the distance between two points is large, SPFLM outperforms PWA* substantially, nevertheless the SPFLM uses 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.

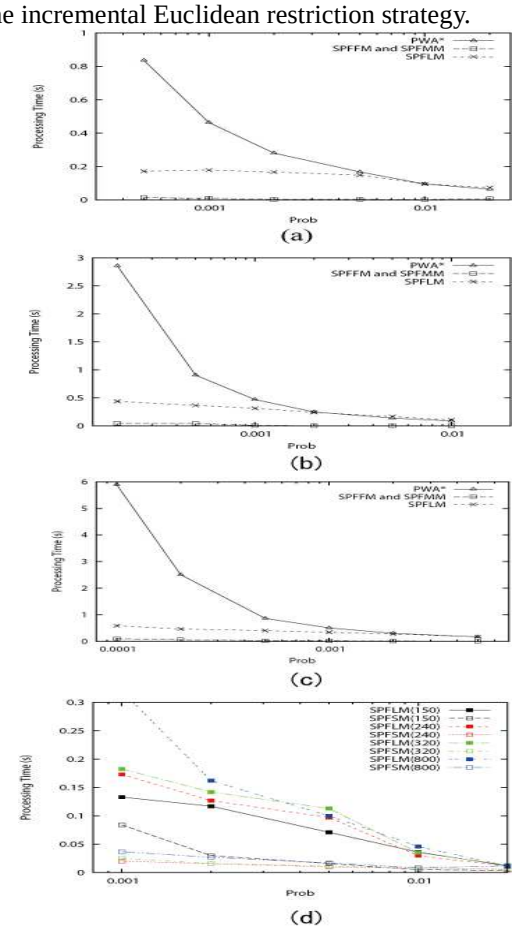


Figure 5. Processing time of the route to 10-NN POIs

In this situation, the relative search speed of PWA* increases, because the hit ratio in LRU buffer managing adjacency list is increased. *k*NN queries evaluated in this paper is for such example. When the density of POI is high, the difference of the

processing times between SPFLM and PWA* becomes small. On the other hand, SPFMM and SPFFM outperform the other methods even in such situation.

References

[1] B.Liu, J.Tay, 11th CAIA, pp.306-312, 1995

[2] L.Fu, D.Sun, L.R.Rilett, Computers & Operations Research 33, pp.3324-3343, 2006

[3] N.Jing et al., 5th ICIKM, pp.268-276, 1996

[4] S.Jung, et al., IEEE Trans. KDE, 14, pp.1029-1046, 2002

[5]S.Shekhar et al., ISLSD, pp.94-111, 1997