PAPER Special Section on Data Engineering and Information Management

Incremental Single-Source Multi-Target A* Algorithm for LBS Based on Road Network Distance

Htoo HTOO[†], ***Student Member***, **Yutaka OHSAWA**[†a)], ***Member***, **Noboru SONEHARA**[††], ***and*** **Masao SAKAUCHI**[††], ***Fellows***

SUMMARY Searching for the shortest paths from a query point to several target points on a road network is an essential operation for several types of queries in location-based services. This search can be performed using Dijkstra's algorithm. Although the A* algorithm is faster than Dijkstra's algorithm for finding the shortest path from a query point to a target point, the A* algorithm is not so fast to find all paths between each point and the query point when several target points are given. In this case, the search areas on road network overlap for each search, and the total number of operations at each node is increased, especially when the number of query points increases. In the present paper, we propose the single-source multi-target A* (SSMTA*) algorithm, which is a multi-target version of the A* algorithm. The SSMTA* algorithm guarantees at most one operation for each road network node, and the searched area on road network is smaller than that of Dijkstra's algorithm. Deng et al. proposed the LBC approach with the same objective. However, several heaps are used to manage the search area on the road network and the contents in each heap must always be kept the same in their method. This operation requires much processing time. Since the proposed method uses only one heap, such content synchronization is not necessary. The present paper demonstrates through empirical evaluations that the proposed method outperforms other similar methods.

key words: *A* algorithm, road network, POI query, incremental Euclidean restriction*

1. Introduction

Calculation of the shortest path and the road network distance between two points is a basic operation in location-based services (LBS). In LBS, target points of a query are points of interest (POIs), which include, for example, restaurants, convenience stores, and gas stations. Usually a search point or multiple search points are specified, and then POIs that satisfy the search conditions, which vary depending on the objective of the queries, are searched. k nearest neighbor (k-NN) queries, aggregate nearest neighbor (ANN) queries, spatial skyline queries, and trip planning queries are examples of these queries. Efficient methods for these queries in Euclidean distance have been investigated for two decades, and such methods for road network distance have also been investigated extensively. Dijkstra's algorithm [1] and the A* algorithm [2] have been used for road network distance calculation.

Manuscript received June 28, 2012.
Manuscript revised October 19, 2012.

[†]The authors are with Saitama University, Saitama-shi, 338–8570 Japan.
[††]The authors are with National Institute of Informatics, Tokyo, 101–8430 Japan.
a) E-mail: ohsawa@mail.saitama-u.ac.jp

DOI: 10.1587/transinf.E96.D.1043

Given a query point q and a set of POIs P, a k-NN query searches a specified number of (k) POIs, which are located closest to q. Papadias et al. [3] proposed two types of algorithms for this query. One is incremental network expansion (INE), which searches neighbor POIs by gradually enlarging the search area centered at a query point q using Dijkstra's algorithm. Enlarging the search area starting from q, POIs on a road network are searched, and the search is terminated when a specified number k of POIs have been found.

The other type of method is incremental Euclidean restriction (IER). The basic idea of this approach is to first search a set of k-NN points C in the Euclidean distance using the R-tree [4], and then to confirm that the points in C are truly k-NN points in the road network distance. Generally, k-NN POIs in the Euclidean distance are not always k-NNs in the road network distance. Hence, by adding the next neighbor in the Euclidean distance to the candidate set, the road network distances of the candidate points are calculated until no more candidates can be a member of the k-NN result.

The road network distance between a pair of points, namely, a query point and a candidate POI, can be calculated using the A* algorithm, which is usually more efficient than Dijkstra's algorithm. We hereinafter refer to this A* algorithm as the pair-wise A* algorithm. However, in A* algorithm, when POIs trend to one side of the query point, the search areas overlap each other. This means that a node is visited several times during a k-NN query. Therefore, the processing time of the pair-wise A* algorithm sometimes exceeds that of the INE (based on Dijkstra's algorithm). In particular, for a high POI density and a large k value, multiple areas overlap, and the efficiency of the pair-wise A* algorithm deteriorates.

The basic operation in the shortest path search is *node expansion*. Node expansion consists of the following steps:

(1) get the node n, which has the minimum cost from the priority queue,
(2) get all nodes adjacent to n using the adjacency list,
(3) calculate the cost of each adjacent node,
(4) compose a record for each node then add the records to the priority queue.

The above described Dijkstra's algorithm and the A* algorithm also perform node expansion.

Ordinarily, since the size of the adjacency list is large, the list is divided into several small blocks. Using this ad-

jacency list, when a record of a node in a block must be referred, the block containing the referring node is read into a least recently used (LRU) buffer, then adjacent nodes are investigated. Although the processing time depends on the hit ratio of the LRU buffer, the processing time of the shortest path search increases almost linearly with the number of expanded nodes.

Deng et al. [5] proposed the lower bound constraint (LBC) approach and the LBC-KNN algorithm for kNN queries, which runs multiple A* algorithms concurrently, and each node is guaranteed to be visited only once. This method was intended to alleviate the problem of pair-wise A* algorithms that can expand a node several times for multiple target points. Although the LBC-KNN is described in detail in Sect. 2.2, the LBC-KNN uses multiple priority queues (PQs) in which included nodes are maintained to be identical. This operation consumes a great deal of processing time.

The present paper proposes another single-source multiple-target A* (SSMTA*) algorithm, which uses only one PQ to overcome the drawback of the LBC-KN. Since the expanded node number is the same as the LBC-KNN, it is lesser in expanded nodes than Dijkstra's and pair-wise A* algorithms. The basic idea of the proposed method has been reported in [6], however, the method proposed in the previous paper has a defect in node expansion by about 30% more than the LBC-KNN. The method proposed herein decreases the number of expanded nodes to be the same as the LBC-KNN. Although, the proposed algorithm can be applied to various searches in the LBS, including k-NN query, ANN query [6], and trip-planning queries [7], the remainder of the present paper focuses on k-NN query in order to show the basic characteristics of the proposed method.

The remainder of the present paper is organized as follows. Section 2 describes the basic characteristics of IER, the A* algorithm, and the LBC-KNN algorithm. In Sect. 3, the SSMTA* algorithm is proposed and applied to k-NN search. Section 4 shows the experimental evaluation results. Finally, Sect. 5 summarizes the present paper and describes future areas for investigation.

2. Preliminary

This section first describes the A* algorithm because the LBC-KNN and the proposed method are based on it. Next, the k-NN search based on IER is described briefly. Finally, the characteristic of the LBC-KNN algorithm, the closest competitor of the proposed method, is summarized.

2.1 A* Algorithm

The targets of the search considered throughout the present paper are points of interest (POIs). However, these POIs are not always existed on a road network node, for simplicity, we assume that POIs are located on a node throughout the present paper. This restriction, however, can be removed [8]. Table 1 summarizes the notation used in the present paper.

Table 1 Notation.

Symbol	Meaning
P	POI set
C	Candidate set for k-NN
p_n	Next POI candidate
q	Query point
PQ	Priority queue
CS	Closed set
$d_E(x, y)$	Euclidean distance between x and y
$d_E^{min}(x, C)$	Minimum Euclidean distance between x and $p \in C$
$d_N(x, y)$	Network distance between x and y
$d_N^k(q, C)$	Network distance between q and k-th NN in C
$RLink(p, q)$	Pointer to the road segment for which the edge nodes are p and q

The A* algorithm searches the shortest path from the origin q to the destination p more efficiently than Dijkstra's algorithm and estimates the cost $Cost$ from q to p via a current node n, by means of the following equation:

$$Cost = d(q, n) + h(n, p)$$

where $d(q, n)$ is the actual cost of moving from q to n on a road network, $h(n, p)$ is the estimated cost between n and p, and the value must be lower than the actual cost. The value returned by $h(n, p)$ is the lower bound of the actual cost, then for any two points a and b, the condition $d(a, b) \geq h(a, b)$ is satisfied. We can assume several types of cost, such as travel distance, travel time, and travel expense, to be minimized. Among these costs, when we minimize the travel distance, the Euclidean distance can be used as the estimated cost. Moreover, when we minimize the travel time, we use the travel time for moving by the expected fastest speed between two points. In the remainder of the explanation, we use the distance on the road network as the cost.

The A* algorithm controls searching by a priority queue PQ. During processing, the following record is composed and inserted into PQ.

$$< Cost, N_C, N_P, d_N(q, N_C), RLink(N_P, N_C) > \quad (1)$$

Figure 1 summarizes the symbols in this record and we used these symbols throughout the rest of the paper with the same meaning. $Cost$ has been described above. N_C is the currently intended node, and N_P is the previously visited node on the path from q to N_C. $d_N(q, N_C)$ is the path length from q to N_C on the road network, and $RLink(N_P, N_C)$ is the pointer to the road link connecting N_P and N_C. Then, $Cost = d_N(q, N_C) + d_E(N_C, p)$, which is the sum of the road network distance from q to N_C and the Euclidean distance between N_C and p.

At the beginning of the search, the following record is inserted into an empty PQ.

$$< d_E(q, p), q, -, 0, - > \quad (2)$$

Here, $-$ denotes the NULL value. When the search begins, N_C is q. Then, no N_P exists. Moreover, $RLink(N_P, N_C)$ does not exist. Then, the NULL value is assigned to these two items. In order to avoid expanding nodes that have already been examined, the once expanded node is inserted into a

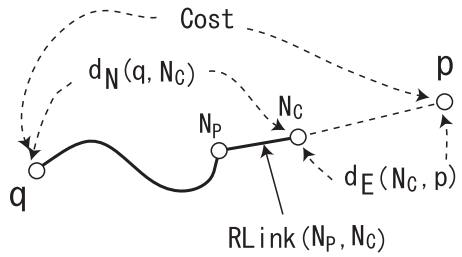


Fig. 1 A* algorithm.

closed set CS, which is initialized by an empty set at the beginning of the search.

The A* algorithm then repeats the following steps until the current node reaches the destination p.

(1) Remove the minimum $Cost$ record from PQ and place this record in CS.
(2) Obtain all nodes neighboring N_C by means of the adjacency list.
(3) For each neighboring node, calculate $Cost$, compose a record shown by Eq. (1), then insert the record with $Cost$ into PQ.

This sequence of steps is a node expansion of the A* algorithm. Among these steps, step (2) needs to access the adjacency list. When the entire adjacency list of the road network is small enough to be stored in the main memory and the adjacency list is read into the buffer in advance, no disk access occurs during the shortest path search. However, the size of the adjacency list is generally large. Therefore, the adjacency list is divided into several small blocks, some of which are read into the LRU buffer, and is then referred to investigate the adjacency nodes. Hence, the number of expanded nodes directly affects the processing time. The A* algorithm is preferred to Dijkstra's algorithm because the former expands a much smaller area of nodes than the latter when a destination is given.

2.2 k-NN Search by IER

The incremental Euclidean restriction (IER) proposed by Papadias et al. [3] searches k-NN by the following steps. Here, the set of POIs is supposed to be indexed by an R-tree [4]. A k-NN search finds number of k nearest neighbor POIs to the query point (q).

(1) Search k-NN in the Euclidean distance using the R-tree, and then store the results in a candidate set C. Subsequently, search k+1 th NN p_n incrementally.
(2) For each point ($p_i \in C$), calculate the road network distance $d_N(q, p_i)$.
(3) If $d_N^k(q, C) \leq d_E(q, p_n)$, then return the top k POIs in C and then stop. Here, $d_N^k(q, C)$ is the road network distance between q and the k-th NN in C.
(4) Calculate $d_N(q, p_n)$. Then, if $d_N^k(q, C) > d_N(q, p_n)$, add p_n to C. Find the next p_n by an incremental search on the R-tree.
(5) Goto Step (3).

In the above algorithm, an incremental nearest neighbor search is required. This search can be performed on an R-tree by a best-first search using a priority queue [9].

When a record is dequeued from the PQ and $d_E(q, p_n) \leq Cost$ stands, the reserved point p_n can become a candidate of k-NN. Then, if multiple A* algorithms can run in parallel, joining p_n to the candidate set C, the searching can be started targeting p_n immediately. In contrast, a pair-wise A* algorithm will not finish until the search reaches the target. Then, if the target locates very far away with respect to the road network distance, the search expends a great deal of processing time and the resulting path may be useless.

2.3 LBC-KNN Algorithm

Deng et al. [5] proposed the LBC-KNN, which can run the A* algorithm in parallel. This algorithm is a simple extension of A* algorithm for multi-targets. The LBC-KNN uses multiple priority queues (heaps), each of which is assigned to a target point. The node to be expanded is obtained from the set of PQs that has the minimum $Cost$ value among the set. Then, the new records composed by the node expansion process are inserted into all PQs. However, the $Cost$ value can differ depending on the target point.

For pair-wise A* algorithm, the expanded areas are overlapped each other when multiple target points are given. Because the same node is expanded repeatedly by the individual pair-wise A* algorithm. On the other hand, for the LBC-KNN, the node expansion is always performed once for each node. Therefore, the LBC-KNN works very efficiently from the viewpoint of the expanded node number. The serious shortcoming of the LBC-KNN, however, is in the synchronization of the nodes contained in the priority queues. After a record that has the minimum cost is removed from a PQ when multiple target points are given, the record must be searched in all other PQs and is removed to keep synchronization of the content. This operation in the set of PQs (Deng et al. [5] uses a set of heaps) has a linear calculation cost and is repeated every time a node is expanded. As such, the calculation cost becomes very high, especially when the number of expanded node is large.

3. SSMTA* Algorithm

This section discusses an efficient multi-target A* algorithm by using a single PQ, which overcomes the shortcoming of the LBC-KNN. First, the basic SSMTA* algorithm is proposed in Sect. 3.1. Next, it is expanded to the incremental version that is capable to kNN query in Sect. 3.3.

3.1 Basic SSMTA* Algorithm

The SSMTA* finds the shortest paths in basically the same way as the A* algorithm. The primary difference is that the SSMTA* searches the shortest paths to multiple target points simultaneously. We hereinafter assume that set C contains all of the target points, and the shortest paths from

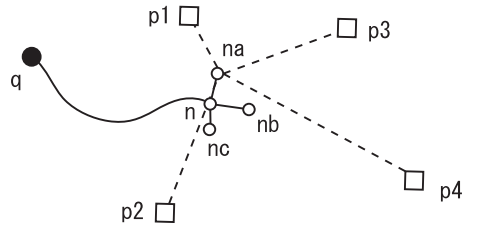


Fig. 2 SSMTA* algorithm.

q to all targets in C are to be calculated. Here, we assume that the target points in C are not changed during the search. In Sect. 3.3, we improve to the case in which new points are incrementally added to C.

Figure 2 shows an example of the node expansion in the SSMTA* algorithm. q is the query point, and four points, from $p1$ to $p4$, are the elements of C. We assume that a path from q to n has already been found, and the record of n is obtained from the PQ to be expanded. Referring to the adjacency list of n, we obtain three nodes from na to nc as directly neighboring n. First, among these nodes, node na is an example. The length of the path from q to na via n is $d_N(q, n) + d_N(n, na)$. For adopting the A* algorithm, we need a heuristic distance from na to the target points in set C. For this, we use the minimum Euclidean distance from na to the points in C, which is denoted as $d_E^{min}(na, C)$. In this figure, the minimum Euclidean distance from na to the points in C is $d_E(na, p1)$, because $p1$ is the nearest point to na in the Euclidean distance. Then, the minimum cost from q to C is $Cost = d_N(q, n) + d_N(n, na) + d_E^{min}(na, C)$. Based on the cost, a new record,

$$< Cost, na, n, d_N(q, n) + d_N(n, na), RLink(n, na) >$$

is created and inserted into the PQ. In this record, the meaning of each item is the same with Eq. (1). The same operation is repeated over the remaining neighbor nodes, nb and nc. This is the node expansion operation used in the SSMTA* algorithm.

At the beginning of the SSMTA* algorithm, the Euclidean distances from q to each point in C are calculated in order to determine $d_E^{min}(q, C)$. The following record is then inserted into the PQ:

$$< d_E^{min}(q, C), q, -, 0, - >$$

Here, $-$ shows the NULL value.

The remainder of the processing is the same as that of the A* algorithm. Since the SSMTA* algorithm treats multiple target points, the algorithm terminates when the search paths reach all of the target points. Algorithm 1 shows the pseudo-code of the SSMTA* algorithm.

In this algorithm, R is the result set of the shortest paths to all target POIs in C. Lines 1 to 3 are initialization code. The result set is initialized by an empty set, the minimum Euclidean distance from q to set C is calculated, and the initial record is composed and inserted into PQ.

The steps beginning from line 4 are repeated until all

Algorithm 1 SSMTA*

Input: C // Candidate POI set
Output: R // Result POI set with the shortest paths

```
 1: R ← ∅
 2: d_min ← min(d_E(q, p_i), p_i ∈ C)
 3: enqueue(< d_min, q, −, 0, − >)
 4: loop
 5:     e ← deleteMin()
 6:     if CS.Contain(e) then
 7:         continue;
 8:     else
 9:         CS.add(< e.N_C, e.N_P, e.d_N, e.RLink >)
10:     end if
11:     if e.N_C ∈ C then
12:         R ← R∪ < e.N_C, getPath(e.N_C) >
13:         C ← C − e.N_C
14:         if |C| = 0 then
15:             return R
16:         end if
17:         RenewQueue(e.N_C)
18:     end if
19:     for all nn ∈ neighbor(e.N_C) do
20:         decide c_i which gives minimum h(nn, c_i)
21:         d_N ← d_N(q, e.N_C) + d_N(e.N_C, nn)
22:         enqueue(< d_N + h(nn, c_i), nn, e.N_C, d_N, RLink(nn, e.N_C >)
23:     end for
24: end loop
```

POIs in C are reached. In line 5, the record e having the lowest $Cost$ value is extracted from PQ. In line 6, whether e already exists in CS is checked. When e is not in CS, e is added to CS.

Line 11 checks whether the search reaches a POI in C. When the search reaches a POI in C, lines 12 through 16 are executed. In line 12, the found POI and the shortest path from q to the POI are registered into the result set R. $getPath(p)$ is the function to restore the shortest path route using CS. In line 13, the found POI is removed from C. As a result, if C becomes empty, all of the shortest paths have been found. Then, the result set R is returned (line 15), and the algorithm is terminated. $RenewQueue(e.N_C)$ in line 17 recalculates the cost in each record in PQ over renewed C. This operation is described in detail later.

Lines 19 through 22 expand each node nn directly neighboring $e.N_C$. Neighboring nodes are found by referring to the adjacency list. The POI (c_i) that gives $d_E^{min}(nn, C)$ is determined, and a PQ entry for nn is then composed and enqueued in the PQ.

RenewQueue in line 17 is the main difference between the original SSMTA* algorithm proposed in [6] and Algorithm 1. This function works as follows. The *wave front* of the node expansion reaches a POI (c_i) in C, and c_i is removed from C. However, the entries in PQ may have the distance to a POI that existed in C in the past, when the cost was calculated, but the POI has since been removed from C. This function then recalculates the cost in PQ based only on the remaining target points. Although the number of records in PQ is not changed by this operation, the order of nodes in PQ may be changed according to the recalculated cost value. In the previously proposed SSMTA* algorithm in [6],

the recalculation operation did not exist. Lack of this operation increases the expanded node number, and requires the modification of the distance of the record in the CS.

RenewQueue operation requires the cost in proportion to the size of PQ. However, PQ contains only wave front nodes and the number of wave front nodes is roughly proportioned to the distance from q. More importantly, this operation is only invoked when C is changed, and it does not require disk access. Consequently, the total required cost in this operation is low. In contrast, the LBC-KNN requires to synchronize the content of PQs every time when a node is expanded. The number of times for this operation is proportional to the square of the distance from q (that means the total number of expanded nodes). The experimental results, which are shown in Sect. 4, exhibit a tendency to drastically increase the processing time according to the increase in the expanded node number.

3.2 Properties of the SSMTA* Algorithm

The following Lemma 1 is the basis for restoring the shortest path from CS entries.

Lemma 1. *For each node N_C on the shortest path from q to p, the value $d_N(q, N_C)$ of a record in CS is the correct shortest path distance from q to N_C.*

Proof. Each record in CS is assigned a provisional network distance $\tilde{d}_N(q, N_C)$ from q to the current node of the record N_C. This means that another shorter path to N_C may exist. However, after once a target POI p is removed from PQ, its $Cost$ value is fixed to $d_N(q, p)$, which is the minimum $Cost$ value in the PQ. In other words, no shorter path can exist. The shortest path between q and p is then fixed.

Let p_P be the previous neighboring node to p on the shortest path. $d_N(q, p)$ was calculated by the equation $d_N(q, p) = d_N(q, p_P) + d_N(p_P, p)$. Here, $d_N(q, p)$ is the shortest distance on the road network. Then, $d_N(q, p_P)$ is also the shortest distance from q to p_P. By repeating this until p_P meets q, $d_N(q, N_C)$ in all records in CS along the shortest path to p is the shortest path length between q and N_C. □

Lemma 2. *Given a destination point set C, the SSMTA* algorithm finds $p(\in C)$ in ascending order of the road network distance.*

Proof. Let two points p and $p'(\in C)$ be considered and let $d_N(q, p) < d_N(q, p')$ be satisfied. Assume that p' is reached in advance of p. Then, just before p' is reached, PQ contains the following two records as the cost value, $d_N(q, n_a) + d_E(n_a, p)$ and $d_N(q, n_b) + d_N(n_b, p')$ (see Fig. 3). Here, n_b is a directly neighboring node to p'. Since p' is visited before p, the condition

$$d_N(q, n_b) + d_N(n_b, p') < d_N(q, n_a) + d_E(n_a, p)$$

holds. However, by the premise, $d_N(q, p) < d_N(q, p')$, and $d_E(n_a, p) \le d_N(n_a, p)$, then

$$d_N(q, n_b) + d_N(n_b, p') < d_N(q, p).$$

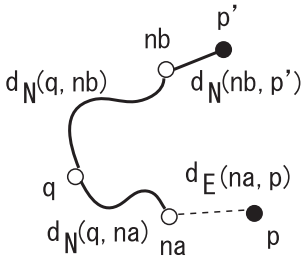


Fig. 3 Explanation of Lemma 2.

This contradicts the hypothesis. Then, p should be reached before p'. This means that the SSMTA* algorithm finds the POI in C in ascending order of the road network distance. □

3.3 Incremental SSMTA* Algorithm

When the SSMTA* algorithm applies to k-NN search by IER, a new target point can be inserted into the candidate set C. This insertion occurs when the $Cost$ of a record dequeued from PQ exceeds the Euclidean distance from q to the reserved POI (p_n) (the reserved candidate is the k+1-th POI in the initial state). At that time, the $Cost$ value of each record in PQ is recalculated over the new C, and the SSMTA* algorithm continues the search until all k-NN have been found.

Algorithm 2 shows a pseudo-code of this incremental SSMTA* (ISSMTA*) algorithm, that finds kNN in the road network distance. j in line 2 counts the number of POIs found shortest paths (see also line 17). EkNN(k, p) searches kNN of q in Euclidean distance, and NextENN(q) in line 4 searches next NN of q in Euclidean distance incrementally. Line 23 checks whether $e.Cost$ exceeds $d_E(q, p_n)$. When the result is true, p_n is added to C, and p_n is replaced by the next NN. In line 26, each $Cost$ value of PQ records is recalculated over modified candidate set.

By this incremental operation, the SSMTA* algorithm is applicable to other types of spatial queries, such as ANN [6] and trip planning queries [7].

3.4 Summary and Contributions

LBC-KNN is a direct extension of the A* algorithm to multi-target search. While SSMTA* algorithm achieves faster search by the following improvements. This subsection summerizes the whole section and describes the contributions of the study.

1. As described in Sect. 2.3, LBC-KNN assigns a PQ set in which each PQ is assigned for each element of the candidate set C. The contents of all PQs must be synchronized to keep the wave front nodes identical among the PQs for different target points. This synchronization is time consuming because it is performed every time a node is expanded.
2. SSMTA* algorithm achieves multi-targets search using

Algorithm 2 ISSMTA*

Input: k,q // k: number of POIs to be found, q: query point
Output: R // Result POI set with the shortest paths

```
1: R ← ∅
2: j ← 0
3: C ←EkNN(k, q)
4: p_n ← NextENN(q)
5: d_min ← min(d_E(q, p_i), p_i ∈ C)
6: enqueue(< d_min, q, −, 0, − >)
7: loop
8:    e ← deleteMin()
9:    if CS.Contain(e) then
10:      continue;
11:   else
12:      CS.add(< e.N_C, e.N_P, e.d_N, e.RLink >)
13:   end if
14:   if e.N_C ∈ C then
15:      R ← R∪ < e.N_C, getPath(e.N_C) >
16:      C ← C − e.N_C
17:      j ← j + 1
18:      if j >= k then
19:         return R
20:      end if
21:      RenewQueue(e.N_C)
22:   end if
23:   if e.Cost > d_E(q, p_n) then
24:      C ← C ∪ p_n
25:      p_n ← NextENN(q)
26:      RenewQueue(e.N_C)
27:   end if
28:   for all nn ∈ neighbor(e.N_C) do
29:      decide c_i which gives minimum h(nn, c_i)
30:      d_N ← d_N(q, e.N_C) + d_N(e.N_C, nn)
31:      enqueue(< d_N + h(nn, c_i), nn, e.N_C, d_N, RLink(nn, e.N_C >)
32:   end for
33: end loop
```

a single PQ. Therefore, it does not require synchronization process that is essential in LBC-KNN. We proposed the basic idea in [6].

3. The algorithm proposed in [6] has a deficit that the original SSMTA* algorithm increases number of expanded nodes in comparison with LBC-KNN. This paper proposes a method to recalculate the $Cost$ value of all records in PQ every time when a target point is deleted from (when the shortest path found) and a new target point is inserted to (in ISSMTA*) the set of target points (C).
4. The correctness of the SSMTA* algorithm controlled by a single PQ is not obvious. Hence, we showed proofs for (1) SSMTA* gives the shortest paths (Lemma 1) and (2) SSMTA* finds the shortest path to the target points in ascending order of the length (Lemma 2). The latter attribute is necessary for kNN search in IER framework.
5. By the above mentioned improvements, the proposed SSMTA* algorithm achieves multi-targets shortest path search by the same expanded node number with LBC-KNN and faster processing time than LBC-KNN, as shown in Sect. 4.

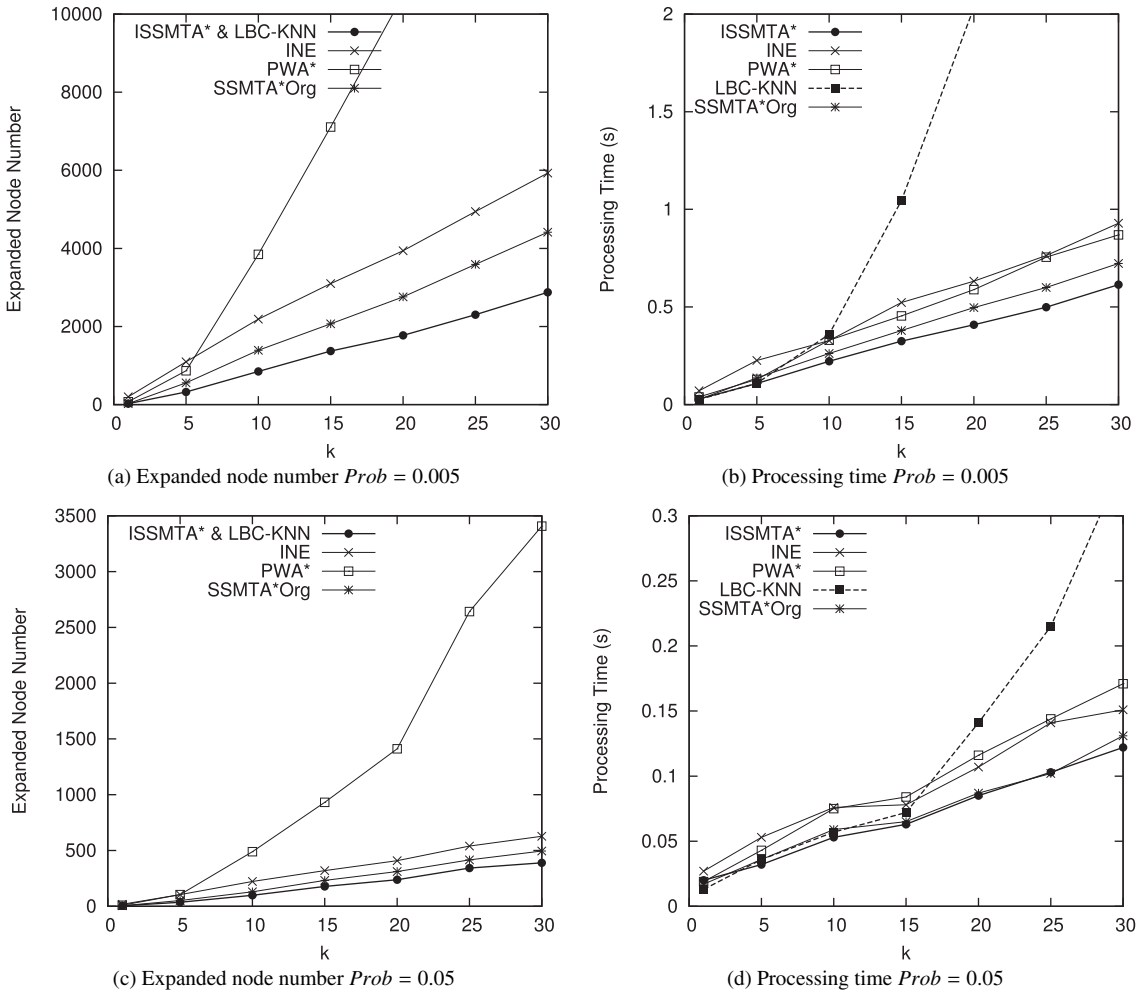
4. Performance Evaluation

This section evaluates the performance of the proposed algorithm by comparison with several conventional methods using a real road network and generated POIs. The road networks used in the experiments are the area of Saitama City, Japan, which has 25,586 road segments (hereafter denoted as "Road-1"), and the area of Saitama Prefecture, which has 468,666 road segments (denoted as "Road-2"). The positions of POIs are generated by a pseudo-random sequence with a specified probability ($Prob$). For example, $Prob = 10^{-3}$ indicates a POI on one thousand road segments. All algorithms are implemented by Java and are evaluated on a PC with an Intel Core i7 CPU 960 (3.2 GHz), 9 GB memory.

The adjacency list used in this experiment is prepared using Peano-Hilbert order as the same with Papadias et al's experiment in [3]. We prepared a 16 KB block adjacency list. The size of the LRU buffer was set to 1 MB (64 blocks). The adapted replacement policy was the popularly used "clock" [10], which acts similar to the LRU. The dividing method of the adjacency list using the Peano-Hilbert order is a simple method. However, the performance is similar to that of Huang's method [11].

Figure 4 compares the k-NN search results among INE, the pairwise A* algorithm (PWA*), the LBC-KNN, the SSMTA* proposed in [6] (SSMTA*org), and ISSMTA* proposed in this paper. Figure 4 (a) and (b) are the result obtained using Road-2 under $Prob = 0.005$. Figure 4 (a) shows the expanded node number. The LBC-KNN and the ISSMTA* show almost the same values. Therefore, both lines overlap each other. When the k value is small (less than 5), the expanded node number in PWA* is small. However, the expanded node number increases more rapidly than in the other methods when k increases. This is because one node is expanded several times during a k-NN search. The vertical axis in Fig. 4 (b) shows the processing time in seconds. The tendency of the increase is almost the same with the expanded node number, except for the LBC-KNN and the PWA*. The processing time of the LBC-KNN increases rapidly when the k value is large. This is because the cost of the heap operation, which is executed every time with each node expansion, increases according to the k value increase. Contrary, the processing time of the PWA* is relatively faster in spite of much expanded nodes number, because the buffer hit ratio of the PWA* is high. The other three methods, INE, SSMTA*org, and ISSMTA*, exhibit stable characteristics. Among them, the ISSMTA* method is the most efficient with respect to both the expanded node number and the processing time.

Figure 4 (c) and (d) show the processing time and the expanded node number in the k-NN search when $Prob$ is 0.05, which is ten times denser than that of Fig. 4 (a) and (b). The expanded node number shows the same tendency except for the LBC-KNN and the PWA* as same as in Fig. 4 (b). In comparing Fig. 4 (d) with (b), the increase of LBC-KNN is



(a) Expanded node number *Prob* = 0.005

(b) Processing time *Prob* = 0.005

(c) Expanded node number *Prob* = 0.05

(d) Processing time *Prob* = 0.05

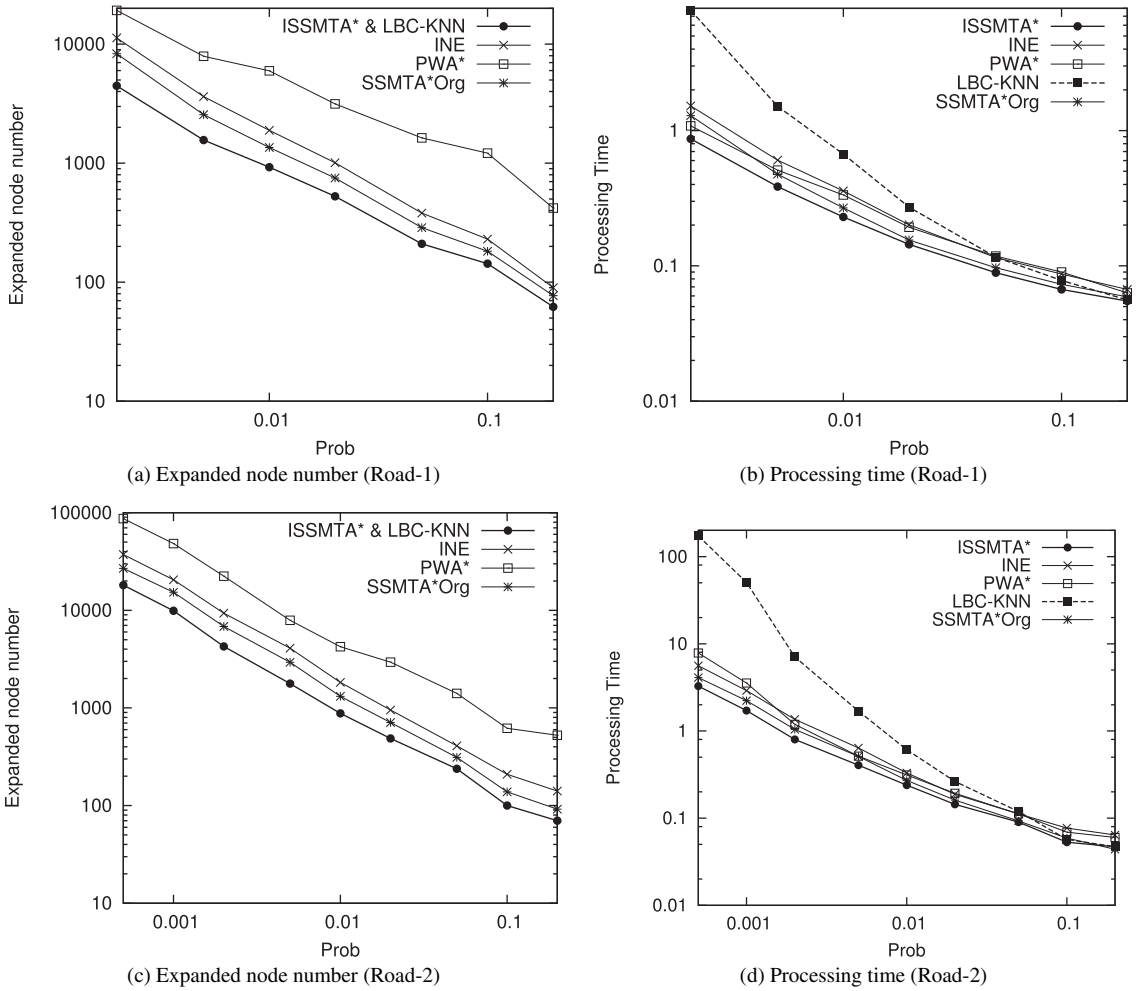
Fig. 4 Comparison methods with k value increase.

alleviated. This is because the total expanded node number decreases to a tenth.

Figure 5 compares the expanded node number and the processing time with varying the density of POI (*Prob*). Figure 5 (a) and (b) are the result on Road-1, and (c) and (d) are the result on Road-2. In this experiment, k value was fixed to 20. The processing time in all methods decreases according to the increase of the POI density because the size of the search area also decreases. Among these methods, the ISSMTA* algorithm is the most efficient in all POI densities. Deng et al. [5] evaluated the performance of the LBC-KNN among the range of $Prob = 0.05$ to 2.0. Over these very high POI density distributions, the search area remains small. Then, the inefficiency of the LBC-KNN does not become apparent. However, when schools, hospitals, convenience stores, and gas stations are considered as POIs, their densities are not so high [12], and such kind of POIs does not exist beyond 0.05. Then, the ISSMTA* algorithm is expected to perform with considerably high efficiency in actual applications in comparison with PWA* and the LBC-KNN.

In this experiment, the results on densities lower than 0.002 for Road-1 are omitted because the entire area on the road network is searched for cases in which $k = 20$. However, the result using Road-2 shows that the proposed method considerably outperforms PWA* and the LBC-KNN in such low-density cases.

When a set of POI is distributed with bias, the result can be predicted as follows; (1) INE requires to search with area on the road network, (2) the areas of expanded node in PWA* become to overlap each other especially near the query point, (3) the expanded node numbers in the LBC and the ISSMTA* are the same, however, the LBC requires much processing time than the ISSMTA* because the average road network distance becomes longer with biased distribution of POIs. Figure 6 shows the result of the experiment to confirm this prediction. The biased sets of POI are prepared by the following method; (1) determine the query point q, (2) generate POIs on the road network nodes that belongs on the right half-plane of q under the same POI density with uniform distribution dealt in the above experiments. Figure 6 compares the expanded node number and the processing time by the ratio against the ISSMTA*. U-time and U-node in this figure show the result on the uniform distribution and B-time and B-node show the result on the



(a) Expanded node number (Road-1)

(b) Processing time (Road-1)

(c) Expanded node number (Road-2)

(d) Processing time (Road-2)

Fig. 5 Evaluation varying POI density (k = 20).

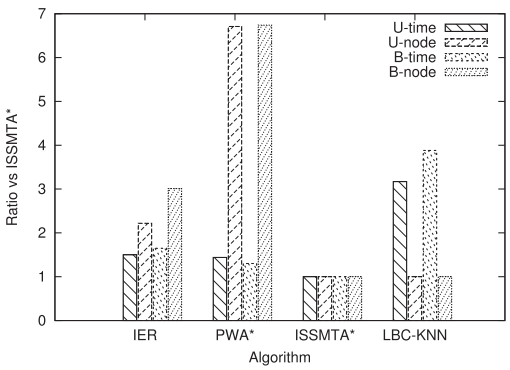


Fig. 6 Comparison between uniform and biased distribution.

biased distribution, respectively. The result meets the above mentioned prediction. As the conclusion, the ISSMTA* algorithm also outperforms the other methods when the set of POI is distributed with bias.

5. Conclusion

We herein proposed the SSMTA* algorithm, which searches shortest paths between a query point and each point in a given destination point set. Then, this algorithm was applied for k-NN queries based on the IER strategy. Through the performance evaluation of the proposed method in comparison with the INE using Dijkstra's algorithm, the pairwise A* algorithm and the LBC-KNN, we showed that the proposed method outperforms the competitors in terms of processing time and expanded node number.

The processing times of the pairwise A* algorithm and the LBC-KNN increases rapidly when the density of POIs is low or the number of k is large. The defects occur for the following reasons. On the pairwise A* algorithm, nodes are expanded several times when k is large, which increases the hard-disk access times. On the LBC-KNN, although the number of node expansions remains low, the cost of PQ scanning increases in proportion to k and the number of node expansions. This performance deterioration is serious when the density of the POI is low.

Although, like the proposed method, Dijkstra's algorithm exhibits stable performance, the expanded node number and processing time remain twice that of the proposed method. A disadvantage of Dijkstra's algorithm is that the performance deteriorates substantially when the distribution of the POIs trends toward one side. This biased distribution is apt to appear in the ANN [8] and spatial skyline queries [5]. Therefore, the ratio of the expanded node number between the INE and SSMTA* becomes larger for these queries [6].

The calculation cost of Dijkstra's algorithm is $O(|E| + |V| \log |V|)$ on the road network $G = (V, E)$. The calculation cost of the A* algorithm varies depending on the heuristic function. However, the cost of searching the shortest path between two specified points is on the same asymptotical order as Dijkstra's algorithm. The worst case occurs when the heuristic function always returns 0. The expanded node number of the LBC-KNN and SSMTA* are the same. This means that the worst time complexity is the same for all of the algorithms considered in the present paper. From the viewpoint of database systems, however, the number of disk accesses dominates the total processing time. Accordingly, the proposed method is at least twice as efficient as the other methods.

As an example of an application of SSMTA*, we applied to ANN queries in [6]. Aside from this, Deng et al. applied the LBC-KNN for spatial skyline queries and obtained positive results. On these types of queries, the POIs to be searched are apt to trend toward one side of the query point. At this situation, the proposed method becomes more efficient than Dijkstra's method. Application of the proposed method to wide spatial queries is a subject of future research.

Acknowledgments

The present study was supported in part by the Japanese Ministry of Education, Science, Sports and Culture (Grant-in-Aid for Scientific Research (C) 24500107) and by the Transdisciplinary Research Integration Center at the Research Organization of Information and Sciences, Japan.

References

[1] E.W. Dijkstra, "A note on two problems in connection with graphs," Numeriche Mathematik, vol.1, pp.269–271, 1959.
[2] P.E. Hart, N.J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," IEEE Trans. Syst. Sci. Cybern., vol.SSC-4, no.2, pp.100–107, 1968.
[3] D. Papadias, J. Zhang, N. Mamoulis, and Y. Tao, "Query processing in spatial network databases," Proc. 29th VLDB, pp.790–801, 2003.
[4] A. Guttman, "R-Trees: a dynamic index structure for spatial searching," Proc. ACM SIGMOD Conference on Management of Data, pp.47–57, 1984.
[5] K. Deng, X. Zhou, H.T. Shen, S. Sadiq, and X. Li, "Instance optimal query processing in spatial networks," The VLDB J., vol.18, no.3, pp.675–693, 2009.
[6] H. Htoo, Y. Ohsawa, and N. Sonehara, "Single-source multi-target A* algorithm for POI queries on road network," WGIM 2011 Workshop, LNCS 7142, pp.51–62, Splinger-Verlag, 2011.
[7] H. Htoo and Y. Ohsawa, "Fast algorithm for simple trip planning queries based on Euclidean restriction," DEIM Forum 2012, pp.D8–1, 2012. (in Japanese).
[8] M.L. Yiu, N. Mamoulis, and D. Papadias, "Aggregate nearest neighbor queries in road networks," IEEE Trans. Knowl. Data Eng., vol.17, no.6, pp.820–833, June 2005.
[9] G.R. Hjaltason and H. Samet, "Distance browsing in spatial databases," ACM Trans. Database Syst., vol.24, no.2, pp.265–318, June 1999.
[10] N. Megiddo and D.S. Modha, "Outperforming LRU with adaptive replacement cache algorithm," Computer, vol.37, no.4, pp.58–65, 2004.
[11] Y.W. Huang, N. Jing, and E.A. Rundensteiner, "Effective graph clustering for path queries in digital map databases," CIKM '96 Proc. Fifth International Conference on Information and Knowledge Management, pp.215–222, 1996.
[12] M. Kolahdouzan and C. Shahabi, "Voronoi-based K nearest neighbor search for spatial network databases," Proc. 30th VLDB, pp.840–851, 2004.

Htoo Htoo received her B.C.Sc and a M.C.Sc degrees from the University of Computer Studies, Yangon, Myanmar in 2002 and 2004. She worked as a teaching staff in University of Computer Studies, Yangon from 2004 to 2008. She is currently a Ph.D student at the Graduate School of Science and Engineering at Saitama University in Japan. Her research interests currently include Geographic Information Systems and spatio-temporal databases.

Yutaka Ohsawa received his B.E. and M.E. degrees from Shinshu University in 1976 and 1978, respectively, and then read for a Ph. D degree at the University of Tokyo in 1985. From 1979 to 1989, he worked for the Institute of Industrial Science at The University of Tokyo as a Research Associate and an Assistant Professor. Since 1989, he has worked for Saitama University. He is currently a Professor at the Graduate School of Science and Engineering at Saitama University. His research interests include Geographic Information Systems, spatio-temporal databases, and intelligent transport systems.

Noboru Sonehara has been a Professor in the Information and Society Research Division at the National Institute of Informatics since 2004. From 2001 to 2004, he was a project manager (Content Commerce Project) at NTT Cyber Solutions Laboratories. He received his BE and ME degrees from Shinshu University, Japan in 1976 and 1978, respectively, and his Ph. D in 1994. He has been a Director of the Information and Society Research Division since 2006. His current research interests include ICT Security, Privacy, Trust, Risk, Resilience, and e-Authentication platforms.

Masao Sakauchi is the Director General of the National Institute of Informatics, Japan, and a Professor in the Institute of Industrial Science at the University of Tokyo. He received his B. Sci degree in electrical engineering from the University of Tokyo in 1969 and his MS and PhD degrees in electronics engineering from the University of Tokyo in 1971 and 1975, respectively. He has acted as the General Chairman and Program Chairman of 10 international conferences and workshops, and has authored more than 450 refereed papers in fields related to multimedia databases.