Fast Computation of Clustered Many-To-Many Shortest Paths and Its Application to Map Matching

GEORGE R. JAGADEESH and THAMBIPILLAI SRIKANTHAN, School of Computer Science and Engineering, Nanyang Technological University, Singapore 639798, {asgeorge, astsrikan}@ntu.edu.sg

Abstract

We examine the problem of computing shortest paths in a transportation network from a set of geographically clustered source nodes to a set of target nodes. Such many-to-many shortest path computations are an essential and computationally expensive part of many emerging applications that involve map matching of imprecise trajectories. Existing solutions to this problem mostly conform to the obvious approach of performing a single-source shortest path computation for each source node. We present an algorithm that exploits the clustered nature of the source nodes. Specifically, we rely on the observation that paths originating from a cluster of nodes typically exit the source region's boundary through a small number of nodes. Evaluations on a real road network show that the proposed algorithm provides a speed-up of several times over the conventional approach when the source nodes are densely clustered in a region. We also demonstrate that the presented technique is capable of substantially accelerating map matching of sparse and noisy trajectories.

CCS Concepts

• **Mathematics of computing~Graph algorithms** • **Theory of computation~Shortest paths** • *Information systems~Spatial-temporal systems*

Additional Key Words and Phrases

Many-to-many shortest paths; map matching; speed-up technique

ACM Reference format:

GEORGE R. JAGADEESH and THAMBIPILLAI SRIKANTHAN. XXXX. Fast Computation of Clustered Many-To-Many Shortest Paths and Its Application to Map Matching. *ACM Trans. Spatial Algorithms Syst.* XXXX, XXXX. XXXX (XXXX XXXX), 20 pages.
DOI: XXXX

1. INTRODUCTION

There are certain situations in which it is necessary to efficiently compute shortest paths in a transportation network from each of a number of source nodes lying within a bounded area to each of a number of target nodes outside it. For instance, a logistics application may need to find the optimal routes from a set of depots in a city to a set of customer locations outside the city in order to assign appropriate vehicles for delivery. The computation of shortest paths originating from a bounded area is also a recurrent step in map matching, which is the process of matching a raw trajectory (i.e. a sequence of inaccurate location measurements) to a path in a road network. Map matching, in turn, is a crucial step in many trajectory-based applications such as travel time estimation [1], route choice modelling [2], route discovery [3] and location prediction [4].

DOI: XXXX

In recent years, there has been an increased interest in map matching sparse and noisy trajectories such as those generated by smartphones using Wi-Fi-based and cellular-network-based positioning technologies [5], [6], [7]. Although such positioning technologies are associated with large positioning errors, they offer some advantages over the standard option of Global Positioning System (GPS) such as high energy efficiency, availability in GPS-denied environments and better privacy protection. A number of successful algorithms based on a variety of approaches have been proposed in the literature for map matching sparse and noisy trajectories (e.g. [8], [9], [10], [11]). They involve the computation of shortest paths from each candidate point inside the error region of a location measurement to each candidate point inside the error region of the subsequent location measurement. In the case of highly noisy trajectories, a large number of candidate points need to be considered for each location measurement, often resulting in an intractable number of shortest path computations [12]. These shortest path computations have been identified by several researchers (e.g. [13], [14]) as the computational bottleneck in map matching algorithms. Our own analysis indicates that, on average, about 90% of the overall computation time in a map matching algorithm is spent on shortest path computations when noisy trajectories are used. Since many applications need to map match streaming trajectories on the fly, there exists a need for accelerating the shortest path computations involved in map matching.

The problem of computing shortest paths along a road network from a set of source nodes within a region to a set of target nodes outside that region is a special case of the many-to-many shortest paths (MSP) problem in which the source nodes are geographically clustered. We refer to this version of the MSP problem as the clustered MSP problem. The conventional way of solving the MSP problem is to perform a single-source shortest path (SSP) computation for each source node. This is typically done by repeated executions of Dijkstra's algorithm [15], which can find the shortest paths from a source node to all the target nodes. However, such a solution, which we refer to as the baseline algorithm in this paper, requires an unacceptably high computation time when the number of source nodes is large [16].

A number of techniques have been proposed in the literature to speed up the computation of the shortest path between two nodes (i.e. the one-to-one shortest path problem), especially in road networks. These include goal-directed techniques that direct the search towards the target node (e.g. [17]) and hierarchical techniques that exploit the natural hierarchy present in road networks (e.g. [18]). A comprehensive overview of such speed-up techniques has been presented in [19]. However, speed-up techniques proposed for the one-to-one case are not directly applicable when dealing with the MSP problem. In general, it is inefficient to transform an MSP problem between a set of source nodes S and a set of target nodes T into $|S| \times |T|$ one-to-one shortest-path computations, where $|S|$ and $|T|$ are the number of source nodes and target nodes, respectively. Even if the one-to-one computations utilize the best speed-up techniques, it is generally still faster to perform $|S|$ executions of Dijkstra-based SSP computations.

While a few papers in the literature have specifically targeted the MSP problem, they have certain limitations as explained in the next section. Motivated primarily by the need for speeding up clustered MSP computations involved in map matching, we present and evaluate an algorithm that is substantially faster than existing solutions when the source nodes are densely clustered in a region. The proposed algorithm identifies a relatively small number of *exit nodes* such that each shortest path originating from any of the source nodes passes through one of the exit nodes before crossing the boundary of the source region. The algorithm uses the exit nodes to subdivide and speed up the MSP computation.

The rest of the paper is organized as follows. The related work on fast MSP computation is described in Section 2. In Section 3, we define the clustered MSP problem and and concisely describe some essential basic concepts. The proposed algorithm for clustered MSP computation is presented in Section 4. In Section 5, we describe the map matching problem and show how the proposed clustered MSP algorithm can be applied to it. Experimental evaluations and results are discussed in Section 6. We summarize and conclude the paper in Section 7.

2. RELATED WORK

The MSP problem has not received much research attention as a separate problem. This is probably because it is generally seen as being composed of multiple instances of the SSP problem, for which Dijkstra's algorithm remains the standard solution. There have been several decades of research efforts dedicated to improving the running time of Dijkstra's algorithm, mainly through the use of efficient data structures to implement the priority queue used by it [19].

Researchers who have specifically tackled the MSP problem have generally adapted speed-up techniques meant for the one-to-one shortest path problem to fit the MSP problem. Knopp et al. [16] proposed a method of applying a hierarchical acceleration technique to the MSP problem. Their basic idea involves performing limited backward searches from each target node on a hierarchically-organized road network and storing the search spaces so that the stored information is accessed during the forward searches from each source node. Delling et al [20] applied an algorithm based on *contraction hierarchies* [18], a state-of-the-art hierarchical acceleration technique, to the MSP problem. While the above algorithms are several orders of magnitude faster than the baseline algorithm for large problem instances, they have a significant drawback: they involve an offline preprocessing step for constructing a hierarchical representation of the road network. The dependence on preprocessing makes it impractical to apply these solutions to road networks in which the connectivity and edge lengths (e.g. travel times) change dynamically. This is because the preprocessing step needs to be repeated, at least partially, whenever there is a change in the road network.

Since the capability to compute shortest paths in a dynamic road network is important for many applications, it is necessary to have solutions that do not involve any preprocessing. One such solution was presented by Shibuya [21], where goal-directed and bidirectional search techniques, usually applied for one-to-one shortest path computations, were adapted for the MSP problem. This method first computes a bounded backward search space from all the target nodes and then uses a lower-bound estimate of the shortest path length to the set of target nodes to accelerate the search from each source node. This produces a reduction in computation time of about 30% compared to the baseline algorithm for normal cases and about 70% when the source nodes and target nodes are located in two distant clusters.

The suitability of goal-directed and bidirectional search techniques for the MSP problem was also investigated by Knopp [22]. It was found that goal-directed search based on the concept of *landmarks* [23] produces the best result (a speed-up of up to four times) for large problem instances when the source nodes and target nodes are confined to two clusters. The landmarks-based MSP algorithm, which we use for comparisons in this paper, can be briefly described as follows: A small number k of source nodes that are farthest from each other are first selected as implicit landmarks. Subsequently, Dijkstra's algorithm is used to find the shortest-path distances from each landmark to all the nodes in the network and vice versa. With these distances known, the triangle inequality can be applied to obtain a lower bound on the shortest-path distance between any node and a target node. This estimate of the distance to the target node is used as a potential function to direct the search towards it. For each source node that is not a landmark, a modified version of Dijkstra's algorithm is invoked to find the shortest paths to all the target nodes, but the search is directed towards one target node at a time. After the shortest path to the current target node is found, a new target node is chosen and the potential function is updated.

A key insight from the above studies is that goal-directed and bidirectional search techniques are effective for the MSP problem only when both the source nodes and the target nodes are clustered. The problem that we consider in this paper accommodates this scenario while allowing a further degree of flexibility: the target nodes need not necessarily be clustered, they may lie anywhere outside the source region. (The converse of this problem, where the target nodes are

confined to a region and the source nodes lie outside it, can be transformed into the original problem by using a reverse graph representation of the road network.)

Some researchers working on the map matching problem have substantially accelerated the bottleneck step of MSP computation by precomputing shortest paths between all pairs of nodes in the road network that lie within a certain distance and storing the results in a hash table [24] [25]. However, as in the case of solutions that require the road network to be preprocessed, the precomputation approach is also not suitable for dynamic road networks.

3. PROBLEM DEFINITION AND PRELIMINARIES

3.1 Problem Definition

Let $G = (V, E)$ be a directed and weighted graph representing a road network with a set of nodes V corresponding to endpoints of road segments and a set of edges $E \subset V \times V$. Each edge $(u, v) \in E$ represents a road segment and has a non-negative length $l(u, v)$. Edge lengths may represent a measure associated with the road segments such as their physical lengths, travel times or tolls. A path in the graph from node u_1 to node u_n is a sequence of nodes $(u_1, u_2, \ldots, u_n)$ such that $(u_i, u_{i+1}) \in E$ for all $i, 0 < i < n$. The length of a path is the sum of the lengths of the edges in it. The shortest-path distance $d(v, w)$ between nodes v and w is the length of the path with the minimum length among all paths between v and w.

Let us consider a set of source nodes $S = \{s_1, s_2, \ldots, s_m\},\ S \subset V$, and a set of target nodes $T = \{t_1, t_2, \ldots, t_n\},\ T \subset V$, such that $S \cap T = \emptyset$. Furthermore, let all the source nodes in S lie within a bounded area A, which we refer to as the source region. The source region A may be of any closed, non-intersecting shape such as a circle, ellipse or a simple polygon. Let all the target nodes in T lie outside the source region A. The clustered MSP problem is to find the lengths of the shortest paths between all pairs of source nodes and target nodes. That is, we need to compute the shortest-path distance $d(s_i, t_j)$ for all $(i, j),\ 0 < i \leq |S|,\ 0 < j \leq |T|$.

We are only interested in computing the lengths of the shortest paths and do not need the details of the sequences of nodes and links that make up the paths. However, if such details are required, it is not difficult to adapt the algorithm to meet the requirement, as explained later in this section.

3.2 Graph Representation

In this work, it is necessary to represent the graph using a data structure that allows efficient computation of shortest paths in both the forward and the reverse directions. For this we use the compact forward and reverse star representation [26], which stores the node adjacency information in the form of arrays. Data associated with nodes and edges are stored in two separate arrays. Nodes and edges in the graph are numbered according to their positions in their respective arrays. While nodes can be stored in any arbitrary order, edges are stored in a specific order: outgoing edges of node 1 are stored first, those of node 2 are stored next, and so on. For each edge, we store its tail node, head node and length. For each node, we store a forward pointer that points to the position of that node's first outgoing edge in the edge array. To facilitate efficient retrieval of the incoming edges of a node, we store a reverse pointer pointing to a position in a lookup array corresponding to the first incoming edge of that node. The lookup array contains the edge numbers sorted according to their head nodes. It is worth noting that the creation of this data structure is a one-time process. In the case of graphs with dynamic edge lengths, the changed lengths can be directly updated in the edge array without altering the other aspects of the data structure. Apart from the graph representation described above, the geographical coordinates of the nodes are stored in a separate array.

3.3 Dijkstra's Algorithm

Dijkstra's algorithm, in its standard form, finds the shortest paths from a given source node s to all other nodes in the graph. In this work, we use a variant of Dijkstra's algorithm that terminates once the shortest-path distances from the source node to all the nodes in a given set of target nodes T are found. (i.e. it finds one-to-many shortest path distances.)

The pseudo code of the modified Dijkstra's algorithm that we use is shown in Algorithm 1. For each node i, the algorithm maintains and updates a tentative distance $dist[i]$, which is an upper bound on $d(s,i)$, the shortest-path distance from s to node i. At the start of the algorithm, $dist[i]$ is set to infinity for each node i. A node i is considered to be *settled* when the algorithm determines that $dist[i]$ cannot be improved further (i.e., $dist[i] = d(s,i)$). The algorithm maintains a minimum-priority queue Q, which is initialized to be empty. When any node i is added to the Q, its *key* is set equal to the value of $dist[i]$ at that time.

The source node s is first added to Q after setting $dist[s]$ to be 0. During each iteration of the *for* loop in line 10, one of the target nodes t is considered as the current target. If t is not already settled, the *while* loop in line 12 is repeatedly executed until a terminating condition is encountered. During each iteration of the while loop, a node u with the lowest key is removed from Q. For each neighbor node v of node u, $dist[v]$ is improved (i.e. reduced) if the path from s to v through u is shorter than the existing value of the tentative distance $dist[v]$. In such a case, node v is added to Q. The while loop terminates when the current target node t is removed from Q and settled. Thus, at the completion of the for loop in line 10, the shortest-path distances from the source node s to all the target nodes in T are found.

ALGORITHM 1. Modified Dijkstra's Algorithm

```
1  Dijkstra (V,E,s,T)
2    Q = ∅
3    for each i ∈ V:
4      dist[i] = ∞
5      settled[i] = false
6
7    dist[s] = 0
8    add s to Q with key equal to dist[s]
9
10   for each t ∈ T:
11     if settled[t] == false:
12       while Q is not empty:
13         remove u with minimum key from Q
14         if settled[u] == false:
15           for each (u,v) ∈ E:
16             if (dist[u] + l(u,v)) < dist[v]:
17               dist[v] = dist[u] + l(u,v)
18               add v to Q with key equal to dist[v]
19           settled[u] = true
20         if u == t:
21           break
```

It is worth noting that unlike commonly used versions of Dijkstra's algorithm, the above version does not use the *decrease-key* operation on the priority queue. That is, when a better value of $dist[v]$ is found for a node v that is already in Q, its key and position in the priority queue are not updated. Instead, the node v is added again to Q with a new key. Doing so does not affect the correctness of the algorithm. We have adopted this approach because it generally achieves better performance compared to using a priority queue with the decrease-key operation [27].

As stated before, the version given in Algorithm 1 computes only the shortest-path distances and does not store the details of the shortest paths. However, if needed, the tracing of the shortest paths can be easily enabled by storing the predecessor node of each node, after the latter's tentative distance is updated in line 17.

4. THE CLUSTERED MSP ALGORITHM

Our solution to the clustered MSP problem emanates from a simple observation: Shortest paths originating from a source region encompassing a large number of nodes typically cross the source region's boundary through a smaller number of nodes. This is illustrated using an example in Figure 1, where the red circle covers a region of 15 km radius containing the road network of Malacca City, a coastal city in Malaysia. While there are nearly 12000 nodes within this region, a shortest path originating from any of those nodes to target nodes outside the region must pass through one of the 37 nodes indicated by the blue dots in the figure. We refer to such nodes as potential exit nodes. An exit node is the last node that lies along a path before the path exits the source region.

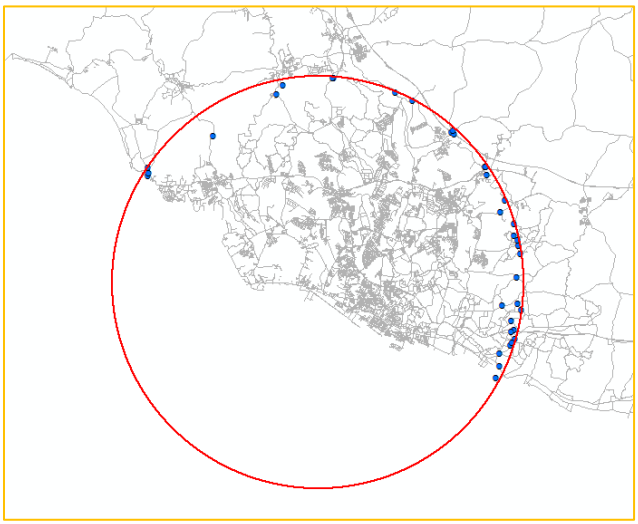

Fig. 1. Potential exit nodes for a source region encompassing the road network of Malacca City in Malaysia. Shortest paths originating from the circular region (15 km radius) cross its boundary through one of the 37 potential exit nodes marked by the blue dots.

In the clustered MSP problem, the shortest path from a source node to a target node can be divided into two parts: the shortest path between the source node and an exit node and the shortest path between that exit node and the target node. Our approach towards solving the clustered MSP problem consists of the following high-level steps:

1. Identify the exit nodes in the source region and compute the shortest-path distances between each exit node and each target node.
2. Compute the shortest-path distances between each source node and each exit node.
3. By optimally combining the distances computed in the above steps, determine the shortest-path distances between each source node and each target node.

The proposed clustered MSP algorithm implements the above three steps in an efficient manner. We give below a detailed and formal description of the algorithm.

As previously defined, let S and T be the sets of source and target nodes, respectively. Let A be the source region encompassing all the source nodes. Let V_A be the set of *all* nodes lying within the source region A. In Step 1, the algorithm determines the set of exit nodes $X = \{x_1, x_2, \dots, x_k\}$. In Step 1 and Step 2, the algorithm incrementally constructs an abstract graph $G' = (V', E')$ with a set of nodes $V' = S \cup T \cup X$ and a set of edges $E' \subset V' \times V'$ such that the length of an edge in G' is equal to the shortest-path distance between its tail node and head node in the original graph G.

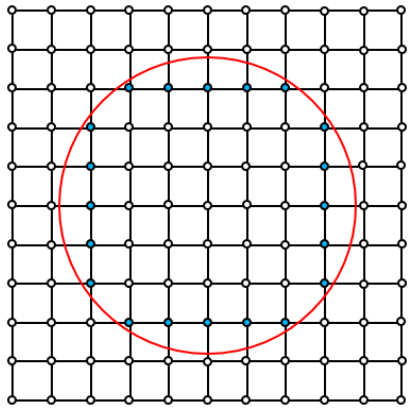

Fig. 2. The concept of potential exit nodes illustrated using a simple grid network. For the source region denoted by the red circle, the blue-shaded nodes are the potential exit nodes.

Step 1: As the exit nodes are not known at the start of the algorithm, we initialize X to be empty. We form a set of all potential exit nodes $P = \{p_1, p_2, \dots, p_r\}$. A node p is considered a potential exit node if there exists an edge $(p, q) \in E$ such that $p \in V_A, q \notin V_A$. In other words, a potential exit node is a node that lies within the source region and has a connected neighbor node that lies outside the source region as shown in the example in Figure 2. It is obvious that any shortest path originating from the source region A to a target node outside it has to pass through one of the potential exit nodes. However, typically, not all of the potential exit nodes are relevant and only a subset of them serve as exit nodes for the given source nodes and target nodes. As a simple example, consider a potential exit node connected to a neighbor node lying to the west of the source region. Such a potential exit node is less likely to be an exit node if all the target nodes lie to the east of the source region. The algorithm identifies the exit nodes as described below.

With each potential exit node $p_i \in P$ as the source, we perform an SSP computation using the version of Dijkstra's algorithm in Algorithm 1 to determine the shortest-path distances to all the target nodes in T. During the SSP computation, for each node u settled by Dijkstra's algorithm, we keep track of the last potential exit node (LPEN) in the shortest path between the source and u. After a node u is settled (line 19 of the pseudocode in Algorithm 1), we update its LPEN value as follows: If u is a potential exit node, $LPEN[u]$ is set equal to u. Else, the previous value of $LPEN[u]$ is retained. Since, in this instance of SSP computation, the source is a potential exit node p_i, all the nodes settled by Dijkstra's algorithm are guaranteed to have a valid LPEN value. The SSP computation terminates when all the target nodes are settled (i.e. the shortest-path distances from p_i to all the target nodes are found). Subsequently, for each target node $t_j \in T$, we check if $LPEN(t_j)$ is equal to p_i. If so, we perform the following actions:

- Identify p_i as an exit node and add it to X, if it is not already in it.
- If p_i and t_j, respectively, are not already in the set of nodes V' in the abstract graph G', add them to V'.
- Create an edge (p_i, t_j) in G' with edge length $l'(p_i, t_j)$ set equal to $d(p_i, t_j)$, the shortest-path distance from p_i to t_j in graph G, computed as described above.

Step 2: In this step, we need to compute the shortest-path distances from each source node $s_i \in S$ to each exit node $x_j \in X$. If $|S| > |X|$, as is generally the case in our problems of interest, it is more efficient to perform $|X|$ SSP computations in the reverse direction. Therefore, with each exit node $x_i \in X$ as the source, we perform a reverse SSP computation using a variant of Dijkstra's algorithm, which explores the *incoming* edges (u, v) of the minimum-distance node u. The reverse SSP computation terminates when all the nodes in S are settled. Subsequently, for each node $s_j \in S$, we perform the following actions:

- If s_j is not already in the set of nodes V', add s_j to V'.
- Create an edge (s_j, x_i) in G' with edge length $l'(s_j, x_i)$ set equal to $d(s_j, x_i)$, the shortest-path distance from s_j to x_i in graph G, found using the reverse SSP computation.

After all the exit nodes in X are processed as described above, the complete abstract graph G' is formed. It is helpful to visualize G' as a layered graph whose nodes are grouped into three sets, S, X and T as illustrated in Figure 3. Each node in S has outgoing edges connecting it to all the nodes in X. Each node in X is connected to one or more nodes in T through outgoing edges. Each node in T has at least one incoming edge emanating from a node in X. The abstract graph G' can be used to find the shortest-path distances from any source node to all the target nodes. (If there is a need to also find the composition of the shortest paths, then for each edge in G', we should store the corresponding path in G as a node sequence.)

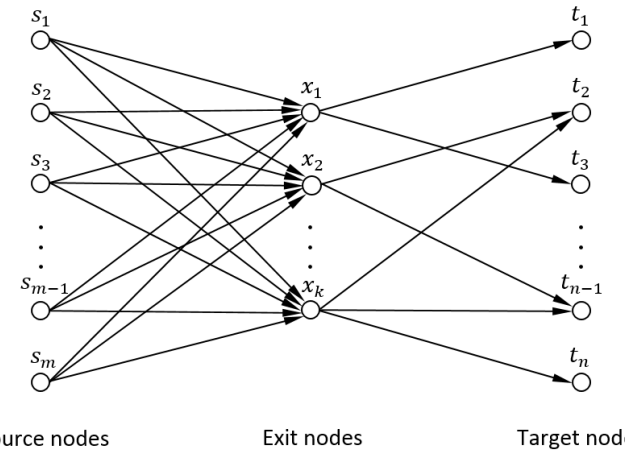

Fig. 3. An abstract graph G′ that represents the connectivity between source nodes and target nodes through exit nodes.

For the sake of clarity, we have omitted a detail in Figure 3 and the preceding description. It is possible that some of the exit nodes are also source nodes. Let us consider one such node x_i that is an exit node as well as a source node. In such a case, x_i, besides being directly connected to at least one target node, will also have outgoing edges connecting it to all other exit nodes. These outgoing edges are relevant only if x_i is the source node from which shortest-path distances to all the target nodes need to be found. On the other hand, if some other source node is being considered, it already has outgoing edges to all the exit nodes with edge weights equal to the

shortest-path distance between them in G. Therefore, in such a case, the outgoing edges from x_i to other exit nodes need not be considered in the shortest path computation. Hence, the graph in Figure 3 can be seen as a fair representation of the connectivity in G'.

Step 3: In this step, we find the shortest-path distances in G' from each source node $s_i \in S$ to each target node $t_j \in T$. This is done by performing an SSP computation for each source node. While the version of Dijkstra's algorithm in Algorithm 1 can be used for this task, it is slightly more efficient to use a customized version that takes advantage of the following observations:

- The shortest path in G' from any source node in S to any target node in T consists of at most two edges. Hence, for any source node s_i, it is sufficient to explore the outgoing edges of s_i and the outgoing edges of all the exit nodes connected to it.
- It is not necessary to maintain a priority queue of nodes and extract the minimum-distance node from it as the order in which the nodes are processed does not affect the final result.

4.1 Computational Complexity Analysis

We express and compare the computational complexities of the conventional and proposed solutions to the clustered MSP problem in terms of the number of SSP computations performed using Dijkstra's algorithm. The running time of the version of Dijkstra's algorithm given in Algorithm 1 is $O\big(|E|\log(|V|)\big)$ in graph $G = (V, E)$ if the priority queue in the algorithm is implemented using a heap [27]. For notational simplicity, we represent this as $O\big(D_{V,E}\big)$, where $D_{V,E} = |E|\log(|V|)$. The baseline algorithm involves $|S|$ SSP computations and has a running time complexity of $O\big(|S| \cdot D_{V,E}\big)$.

In Step 1 of the proposed clustered MSP algorithm, $|P|$ SSP computations are performed in graph G. In Step 2, $|X|$ reverse SSP computations are carried out in G. In Step 3, $|S|$ SSP computations are performed in the abstract graph G'. However, since G' is typically much smaller compared to G and shortest paths in it involve at most two edges, the total computation time of Step 3 is negligible compared to Step 1 and Step 2. Therefore, we do not consider Step 3 in our complexity analysis. Hence, the proposed clustered MSP algorithm involves a total of $(|P| + |X|)$ SSP computations in G. It needs to be noted that $X \subseteq P$, which means, in the worst case, $|X| = |P|$. Therefore, the running time of the clustered MSP algorithm is $O\big(|P| \cdot D_{V,E}\big)$. Based on the above, the speed-up provided by the proposed algorithm over the baseline algorithm is $O(|S|/|P|)$. It follows that the proposed algorithm is not effective for all possible values of $|S|$ and $|P|$. For instance, the running time of the proposed algorithm will be slower than the baseline algorithm for cases where $|S| < |P|$. However, for problems that we are interested in (e.g. map matching), $|S|$ is generally greater than $|P|$.

Without loss of generality, let us consider a circular source region A with radius R. If the spatial distribution of the nodes in the network is assumed be even, the number of potential exit nodes $|P|$ is proportional to the circumference of the source region. It trivially follows that $|P|$ is linearly related to R. Hence, the worst-case time complexity of the clustered MSP algorithm can be expressed as $O\big(R \cdot D_{V,E}\big)$. From the above, it can be seen that the speed-up provided by the proposed algorithm over the baseline algorithm increases with the value of $\frac{|S|}{R}$. In other words, the speed-up increases with the density of the source nodes in the source region.

5. Application to Map Matching

The basic aim of map matching is to match a sequence of imprecise location measurements $o_1, \ldots, o_N$ (i.e. a trajectory) to a sequence of road segments $r_1, \ldots, r_N$ in a road network. A location measurement is a tuple consisting of a latitude, longitude and timestamp. For map matching sparse and noisy trajectories, probabilistic sequence models such as Hidden Markov Model (HMM) [8], [28], [29] and Conditional Random Fields (CRF) [11], [30], [31] are widely used. For the experiments in this paper, we use a HMM-based map matching algorithm that we have previously developed [32]. It is briefly described below.

Each imprecise location measurement in the trajectory is considered as an *observation* in the HMM. Due to the measurement error, the true on-road point corresponding to an observation is unknown. For each location measurement obtained at a particular time step, the closest point on each road segment that lies within a certain error range is considered as a candidate point. Each candidate point is represented as a state in the HMM. All the states corresponding to a time step form a stage in the trellis representation of the HMM. The above concepts are illustrated using a simplified example in Figure 4 (top), where the grid represents a network of road segments. The black dots denote three location measurements (i.e. observations) o_1, o_2 and o_3 obtained at time steps 1, 2, and 3, respectively. The dashed circle around each location measurement indicates its error region. The black squares denote candidate points (i.e. states) on the road segments lying within the error regions. The k^{th} state at time step t is denoted as $c_{t,k}$. The HMM trellis corresponding to this example is shown in Figure 4 (bottom), where the arrows indicate transitions between states.

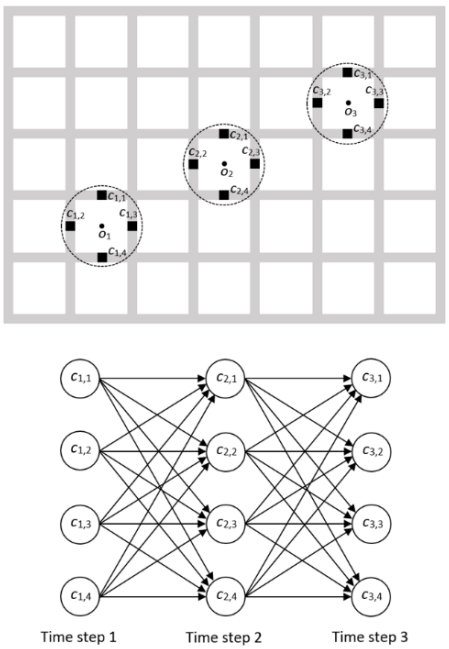

Fig. 4. (Top) A simplified example showing location measurements and candidate points on a grid representing a road network. (Bottom) The HMM trellis representation.

For each state, an emission probability is computed as a Gaussian function of the distance between itself and the corresponding location observation. For each pair of states belonging to successive stages in the HMM trellis, a transition probability is assigned. The transition probability between any two states depends on the nature of the shortest path between them, specifically its circuitousness and the plausibility of traversing it within the elapsed time between

the two corresponding time steps. The most likely sequence of states in the HMM trellis is the sequence that maximizes the product of the emission and transition probabilities along it. An online variant of the Viterbi algorithm is used for finding the most likely sequence of states. A more detailed description of the map matching algorithm can be found in Section II of our previous paper [32].

It can be seen from the above description that the computation of transition probabilities in the HMM requires the shortest-path distances to be computed from each candidate point at one time step to each candidate point at the next time step. That is, at each time step, an MSP computation is required between two clusters of candidate points bounded by the error regions of the corresponding location measurements. When noisy positioning technologies such as cellular positioning are used, the error regions are large with each of them typically containing several hundred candidate points. In such a situation, the amount of time required for computing the shortest-path distances becomes prohibitively large, especially for applications where a large number of trajectories need to be map matched in real time. We intend to accelerate this computation using the proposed clustered MSP algorithm. However, there is an issue to be resolved first: In some cases, especially when the error regions are large and the sampling interval is small, successive error regions may overlap. The clustered MSP algorithm, which assumes that all the target nodes lie outside the source region, cannot be directly applied in such cases. We handle this situation in the following manner.

Let us consider two location observations o_t and o_{t+1} obtained at time steps t and $t+1$, respectively. Let R be the error range of the positioning technology used. Let A_t and A_{t+1} be two circular error regions of radius R with o_t and o_{t+1} as their respective centers. If the circular regions A_t and A_{t+1} do not overlap each other, we directly apply the proposed clustered MSP algorithm by considering A_t as the source region, all the candidate points of o_t as the set of source nodes S, and all the candidate points of o_{t+1} as the set of target nodes T. An example where the regions A_t and A_{t+1} partially overlap each other is shown in Figure 5, where the red squares and the green triangles indicate the candidate points of o_t and o_{t+1}, respectively. In such a case, we form a set S' consisting of the candidate points of o_t lying in the non-overlapped portion of the region A_t, shaded red in the figure. We also form a set S'' consisting of the candidate points of o_t lying in the overlapped portion of the region A_t, shaded brown in the figure. As before, set T consists of all the candidate points of o_{t+1}. We apply the clustered MSP algorithm to find the shortest-path distances from each node in S' to each node in T. We apply the baseline algorithm for finding the shortest-path distances from each node in S'' to each node in T.

For the sake of clarity, the above explanation relies on a simplification that is not strictly true: candidate points are considered as source nodes and target nodes in the MSP computations. However, candidate points, which lie on road segments, may not always coincide with nodes, which are endpoints of road segments. The MSP computations in this work are actually performed using nodes in the road network as source/target nodes. Subsequently, the shortest-path distances computed by the algorithm are offset to account for the distance between the candidate points and the nodes used.

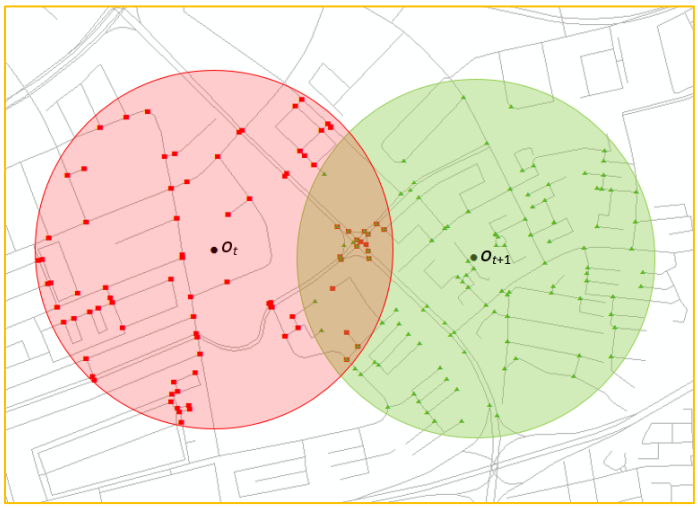

Fig. 5. When map matching noisy trajectories, error regions of successive location observations may overlap each other. In the above example, the circular error regions centered on the location observations o_t and o_{t+1}, respectively, are the source and target regions for the MSP computation. The clustered MSP algorithm is applied only for the source nodes in the non-overlapped portion (shaded red) of the source region.

6. Evaluation

We evaluate the clustered MSP algorithm in two stages. In the first stage, we evaluate the performance of the algorithm in a standalone manner. In the second stage, we embed the clustered MSP algorithm as a subroutine in the HMM-based map matching algorithm, invoke it as described in the previous section and assess the improvement in the running time.

6.1 Standalone Evaluation

For the standalone evaluation of the clustered MSP algorithm, we use a road network of Western Malaysia and Singapore, consisting of about 1.38 million directed road segments, derived from OpenStreetMap map data. The evaluations are performed on a HP Z420 Workstation with an Intel Xeon E5-1650 v2 (3.5 GHz) processor and 16 GB physical memory. We study the speed-up provided by the clustered MSP algorithm over the baseline algorithm in terms of running time as well as the total number of SSP computations performed. The running time speed-up is the ratio of the running time of the baseline algorithm to that of the clustered MSP algorithm. Similarly, we define the SSP computation speed-up as the ratio of the total number of SSP computations performed in the baseline algorithm to that of the clustered MSP algorithm.

We randomly generate a number of instances of the clustered MSP problem. Each problem instance is defined by specifying the radius of the source region R and the number of source nodes $|S|$. In all the standalone experiments, we consider the number of target nodes $|T|$ to be equal to $|S|$. For each problem instance, a source region of radius R is formed centered on a randomly-chosen point in the road network. Subsequently, the specified number of source nodes are randomly chosen from among the nodes lying within the source region. (If the source region does not contain the required number of nodes, it is discarded and the above steps are repeated.) The same number of target nodes are randomly chosen from among nodes lying outside the source region. A $|S| \times |T|$ array is used to store the output of the MSP computation (i.e. lengths of the shortest paths from each source node to each target node). In all the cases, the outputs produced by the proposed and baseline algorithms are identical.

We consider five different values of R, ranging from 1000 m to 5000 m. For each value of R, $|S|$ is varied such that $|S|/R$, which indicates the density of the source nodes in the source region, ranges from 0.1 to 0.5, in steps of 0.1. (For example, for a source region radius of 1000 m, the number of source nodes is kept at 100, 200, 300, 400 and 500.) Thus, a total of 25 different combinations of R and $|S|$ are considered. For each combination of R and $|S|$, 30 instances of the clustered MSP problem are randomly generated as mentioned above and solved using both the compared algorithms. Figure 6 shows the average running time speed-ups for all values of R and $|S|$ considered. The average speed-up provided by the clustered MSP algorithm over the baseline algorithm ranges from 3 times (R = 1000m, $|S|/R$ = 0.1) to 20 times (R = 5000m, $|S|/R$ = 0.5). For every value of R, increasing the density of source nodes (i.e. $|S|/R$) produces a consistent increase in the running time speed-up. This indicates that the proposed algorithm is well-suited for MSP computations in situations where the source nodes are densely clustered.

Figure 7 shows the speed-up in terms of the total number of SSP computations. While the SSP computation speed-ups follow the same trend as the running time speed-ups, the former are lower compared to the latter. To understand the reason for this difference, we need to recall that while the baseline algorithm involves $|S|$ SSP computations, the clustered MSP algorithm performs $(|P| + |X|)$ SSP computations. Therefore, the SSP computation speed-up is equal to $|S|/(|P| + |X|)$. However, the $|X|$ SSP computations performed in Step 2 of the proposed algorithm generally explore a much smaller search space compared to the other SSP computations. This is because each SSP computation in Step 2 is carried out between an exit node and the source nodes, all of which lie within the source region. Therefore, while the inclusion of $|X|$ in the calculation of SSP computation speed-up causes the speed-up values to be lower, the $|X|$ SSP computations do not significantly impact the total running time.

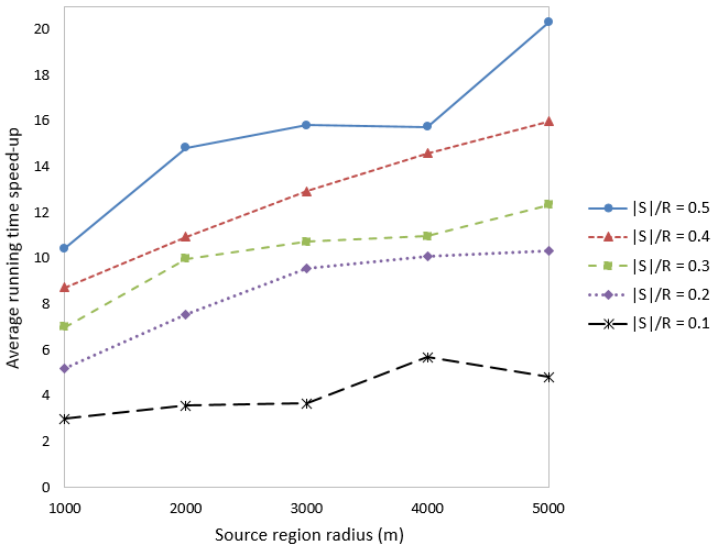

Fig. 6. The average running time speed-ups provided by the Clustered MSP algorithm over the baseline algorithm.

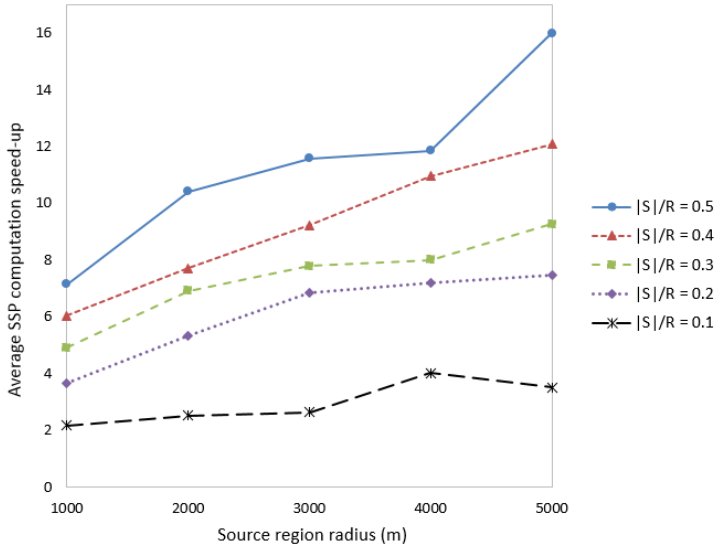

Fig. 7. The average SSP computation speed-ups provided by the Clustered MSP algorithm over the baseline algorithm.

As noted in Section 4.1, if the nodes in the road network are evenly distributed, the number of potential exit nodes $|P|$ in a source region grows linearly with the region's radius R. Based on this, we may expect the speed-up to remain roughly constant for a given value of $|S|/R$. However, in Figure 6 and Figure 7, the speed-up generally increases with R, even as $|S|/R$ remains the same. This is because, in reality, $|P|$ and R do not appear to have a linear relationship. The box plot in Figure 8 shows the distribution of $|P|$ for different values of R (based on the 25 × 30 problem instances), with the horizontal line inside each box indicating the median of that distribution. It can be seen that the median value of $|P|$ grows sublinearly with R. (i.e. At higher values of R, $|P|$ values are generally lower than what they would be if $|P|$ and R were linearly related.) This leads to higher speed-ups at higher values of R because the speed-up provided by the proposed algorithm is inversely related to $|P|$.

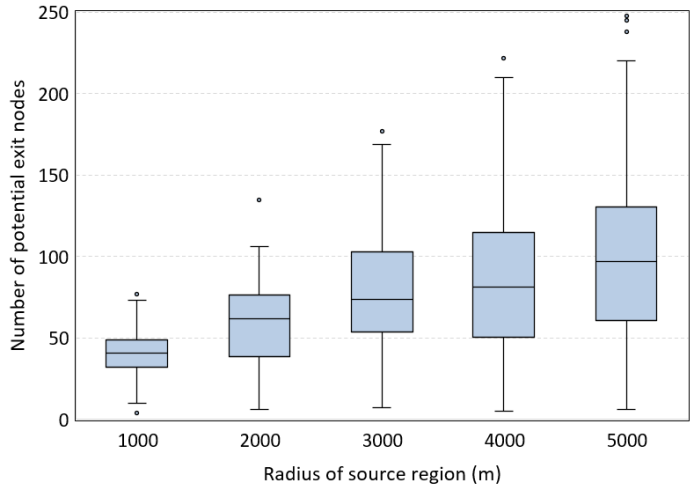

Fig. 8. The distribution of the number of potential exit nodes |P| for source regions with different radii.

Figure 9 shows a scatter plot where the running-time speed-ups observed for all the problem instances are plotted against the corresponding values of $|S|/|P|$. The graph indicates that the running time speed-up is broadly proportional to $|S|/|P|$, thus confirming the analysis presented in Section 4.1. In all the problem instances considered in this experiment, the value of $|S|/|P|$ is greater than 1 and the proposed clustered MSP algorithm is faster than the baseline algorithm.

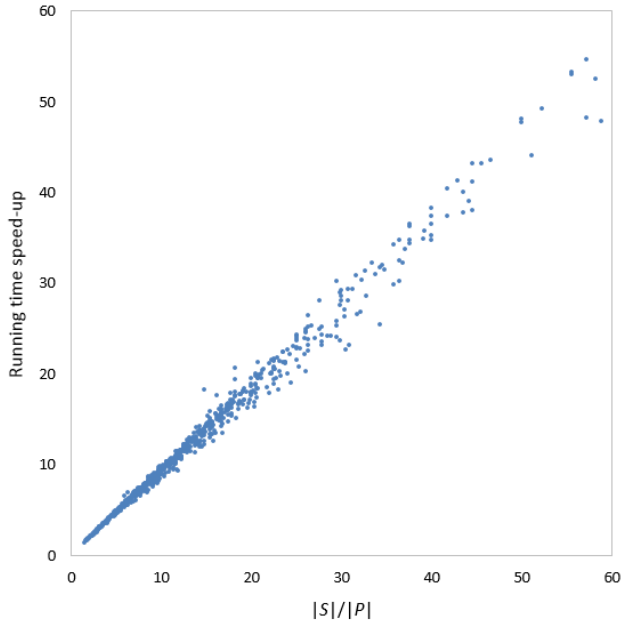

Fig. 9. The relationship between the running time speed-up provided by the clustered MSP algorithm and the ratio of the number of source nodes to the number of potential exit nodes.

Besides the above experimental analysis, we also study the effectiveness of the clustered MSP algorithm in a more practical situation, namely, the computation of shortest-path-distances from a set of source nodes within a city to a set of target nodes outside it. We consider a circular region of 15 km radius encompassing Malacca City (shown in Figure 1) as the source region. For this source region, the number of potential exit nodes $|P|$ is 37. We consider 4 different values of the number of source nodes $|S|$: 1, 10, 100 and 1000. For each value of $|S|$, 30 instances of the clustered MSP problem are randomly generated with all the source nodes in each problem instance lying within the above source region. Figure 10 shows the average running times of the compared algorithms for different values of $|S|$ as a log-log graph. While the running time of the baseline algorithm increases roughly proportionally with the number of source nodes, the running time of the proposed algorithm remains mostly independent of it. For cases where $|S| < |P|$, (i.e. $|S| <$ 37 in this experiment), the baseline algorithm outperforms the proposed algorithm. The converse is true for cases where $|S| > |P|$. It can be inferred from the graph that the speed-up provided by the clustered MSP algorithm over the baseline algorithm is broadly proportional to |S|/|P|. A speed-up of 25 times is achieved for the case where |S| = 1000. The proposed algorithm provides substantially higher speed-ups when isolated regions sparsely connected to the rest of the road network are considered as source regions. For instance, in the case of Penang Island, which is connected to mainland Malaysia through two bridges (i.e. it has only two potential exit nodes), the speed-ups observed are an order of magnitude higher than those in the case of Malacca City.

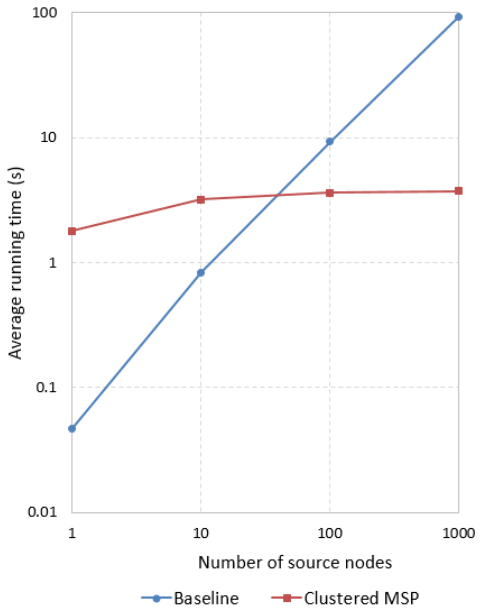

Fig. 10. Comparison of the average running times of the clustered MSP algorithm and the baseline algorithm for different numbers of source nodes within a circular region of 15 km radius with 37 potential exit nodes.

6.2 The Impact On Map Matching Performance

We compare the running times of three versions of the map matching algorithm described in Section 5. While all the three versions are functionally equivalent (i.e., their outputs are identical), each version uses a different algorithm as a subroutine for the MSP computations between successive sets of candidate points. The three algorithms used for the MSP computations are the baseline algorithm, the landmarks-based MSP algorithm [22] and the proposed clustered MSP algorithm. It is worth noting that unlike in the standalone evaluations where the target nodes are randomly distributed outside the source region, the target nodes are clustered within a bounded area in the case of map matching. As seen in Section 2, goal-directed approaches such as the landmarks-based MSP algorithm are known to provide a speed-up over the baseline algorithm when the target nodes are clustered. Hence, we include the landmarks-based MSP algorithm in our comparative evaluation.

For evaluating the three versions of the map matching algorithm under different levels of measurement noise, we use the following trajectory datasets:

- Cellular positioning dataset: This dataset consists of 20 imprecise trajectories generated using cellular network positioning. These trajectories were recorded using a smartphone during taxi trips made in Singapore. The average length and duration of the trips are 21 km and 24 minutes, respectively. The location measurement error of these trajectories has a standard deviation of 382 m. The ground truth paths corresponding to the noisy trajectories were constructed using accurate GPS trajectories that were simultaneously recorded using another smartphone.
- Synthetic dataset (100 m noise): This dataset was created by adding random gaussian noise with a standard deviation of 100 m to 30 ground truth trajectories corresponding to taxi trips in Singapore. The average length and duration of the trips are 18 km and 22 minutes, respectively.
- Synthetic dataset (200 m noise): This dataset is the same as the above except that the added gaussian noise has a standard deviation of 200 m.

While the trajectories were originally recorded at the rate of one sample per second, for the evaluations, we down-sampled them at different rates with the sampling interval ranging from 1 to 5 minutes.

In the map matching algorithm, the error range of the location measurements is set to be 3 times the standard deviation of the location measurement error. Irrespective of which algorithm is used as a subroutine for the MSP computations, the map matching accuracy (the degree of agreement between the map matched path and the ground truth path) remains the same. However, our focus here is not on the map matching accuracy, but on the running time of the map matching algorithm.

Figure 11, Figure 12 and Figure 13 show the average running times taken by the three versions of the map matching algorithm for processing one trajectory in each of the three datasets. The running times of all the versions are higher at lower sampling intervals because, in such cases, a higher number of location observations need to be processed, resulting in a higher number of MSP computations. For all the trajectory datasets and for all the sampling intervals considered, the version with the clustered MSP algorithm outperforms the other two versions by a significant margin. The landmarks-based MSP algorithm performs considerably worse than the baseline algorithm at 1-minute sampling interval, but the former's performance improves at higher sampling intervals. This indicates that the effectiveness of landmarks-based MSP algorithm, which performs goal-directed search, increases with the separation between the clusters of source nodes and target nodes. (The sampling interval generally correlates with the distance between centers of the source and target regions in the MSP computation.) The speed-up provided by the clustered MSP algorithm increases with the sampling interval for all the datasets. This is not surprising as, at low sampling intervals, it is more likely for the source and target regions to overlap each other, thereby reducing the usage of the clustered MSP algorithm. When cellular positioning trajectories are sparsely sampled at the rate of 1 location measurement every 5 minutes, we observe that the clustered MSP algorithm produces a 82% reduction (i.e. a speed-up of 5.6 times) in the overall running time of the map matching algorithm.

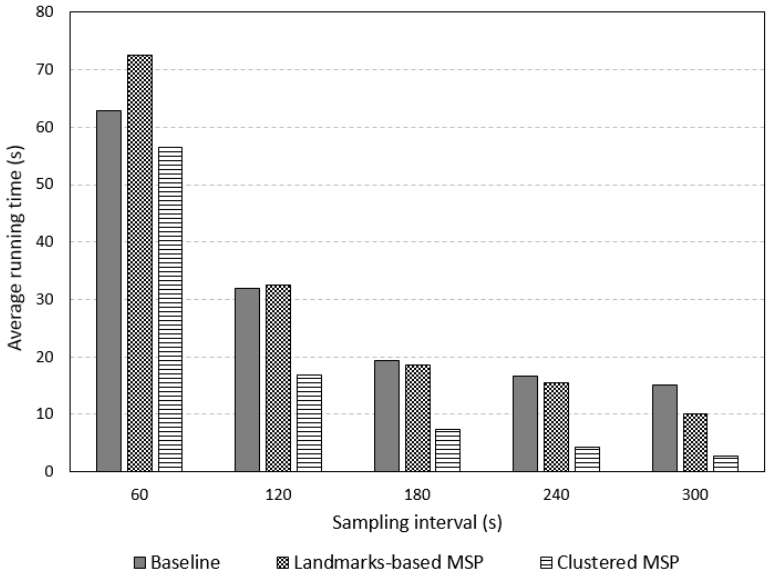

Fig. 11. Comparison of the average running times of three versions of the map matching algorithm for the cellular positioning dataset.

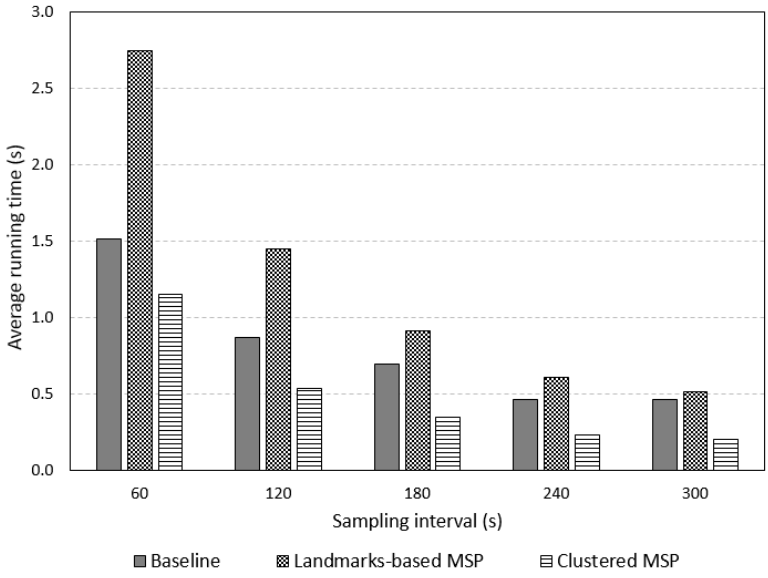

Fig. 12. Comparison of the average running times of three versions of the map matching algorithm for the synthetic dataset (100 m noise).

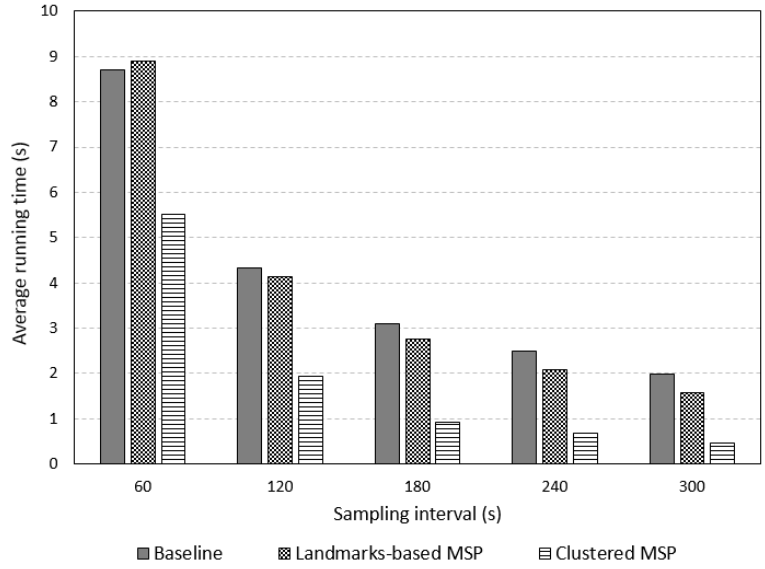

Fig. 13. Comparison of the average running times of three versions of the map matching algorithm for the synthetic dataset (200 m noise).

7. Conclusions

We have presented an alternative approach for efficiently computing shortest paths from a cluster of source nodes in a road network to a set of target nodes. The approach involves identifying a set of exit nodes in the source region that lie along the shortest paths from the source nodes to the target nodes. The many-to-many shortest path computation is accelerated by breaking it

down into two stages: computing shortest paths between the exit nodes and the target nodes and doing the same between the source nodes and the exit nodes. An abstract graph representation is used to optimally combine the results of the above computations. Unlike many existing techniques that seek to accelerate shortest path computations by preprocessing the road network, the proposed solution can be applied to networks with dynamically changing connectivity and edge lengths.

Experimental evaluations using a large number of random problem instances on a real road network show that the proposed clustered MSP algorithm achieves a substantial speed-up over the conventional solution, with the speed-up increasing with the density of source nodes in the source region. We have also successfully applied the clustered MSP algorithm for accelerating the computationally expensive clustered MSP computations involved in map matching of imprecise trajectories. We find that in the case of sparse trajectories with high levels of noise, the proposed algorithm achieves a major reduction in the overall computation time.

The proposed solution can be potentially optimized further by combining it with standard speed-up techniques such as goal-directed search, especially when the target nodes are clustered in a region. Another possible way of extending this work is to explore ways to heuristically limit the selection of potential exit nodes in Step 1 of the algorithm. The technique presented in this paper can be potentially used for accelerating a wider class of shortest path problems in various practical applications. For instance, it appears feasible and interesting to transform the computation of all pairs shortest paths (APSP) in a large network into a number of smaller problems by partitioning the network into several regions and performing one APSP computation and one clustered MSP computation for each region. It would also be interesting to study the applicability of the proposed technique for spatial networks other than road networks.

References

[1] Erik Jenelius, and Harris N. Koutsopoulos, 2013. Travel time estimation for urban road networks using low frequency probe vehicle data. *Transportation Research Part B: Methodological*, 53, pp.64-81.

[2] Nagendra Dhakar and Sivaramakrishnan Srinivasan, 2014. Route choice modeling using GPS-based travel surveys. *Transportation Research Record: Journal of the Transportation Research Board*, (2413), pp.65-73.

[3] Jing Yuan, Yu Zheng, Xing Xie, and Guangzhong Sun, 2013. T-drive: Enhancing driving directions with taxi drivers' intelligence. *IEEE Transactions on Knowledge and Data Engineering*, 25(1), pp.220-232.

[4] Jochen Eisner, Stefan Funke, Andre Herbst, Andreas Spillner, and Sabine Storandt, 2011. Algorithms for matching and predicting trajectories. In *Proceedings of the Thirteenth Workshop on Algorithm Engineering and Experiments (ALENEX)*. Society for Industrial and Applied Mathematics, pp. 84-95.

[5] Arvind Thiagarajan, Lenin Ravindranath, Hari Balakrishnan, Samuel Madden, and Lewis Girod, 2011. Accurate, Low-Energy Trajectory Mapping for Mobile Devices. In *8th USENIX Conference on Networked Systems Design and Implementation (NSDI)*.

[6] Reham Mohamed, Heba Aly, and Moustafa Youssef, 2014. Accurate and efficient map matching for challenging environments. In *Proceedings of the 22nd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. ACM, pp. 401-404.

[7] Gunnar Schulze, Christopher Horn, and Roman Kern, 2015. Map-matching cell phone trajectories of low spatial and temporal accuracy. In *18th IEEE Intelligent Transportation Systems Conference (ITSC)*, pp. 2707-2714.

[8] Paul Newson, and John Krumm, 2009. Hidden Markov map matching through noise and sparseness. In *Proceedings of the 17th ACM SIGSPATIAL international conference on advances in geographic information systems*. ACM, pp. 336-343.

[9] Mahmood Rahmani and Haris N. Koutsopoulos, 2013. Path inference from sparse floating car data for urban networks. *Transportation Research Part C: Emerging Technologies*, 30, pp. 41-54.

[10] Michel Bierlaire, Jingmin Chen, and Jeffrey Newman, 2013. A probabilistic map matching method for smartphone GPS data. *Transportation Research Part C: Emerging Technologies*, 26, pp. 78-98.

[11] Timothy Hunter, Pieter Abbeel, and Alexandre Bayen, 2014. The path inference filter: model-based low-latency map matching of probe vehicle data. *IEEE Transactions on Intelligent Transportation Systems*, *15* (2), pp. 507-529.

[12] Hannes Koller, Peter Widhalm, Melitta Dragaschnig, and Anita Graser, 2015. Fast hidden Markov model map-matching for sparse and noisy trajectories. In *18th IEEE Intelligent Transportation Systems Conference (ITSC)*, pp. 2557–2561.

[13] Yin Lou, Chengyang Zhang, Yu Zheng, Xing Xie, Wei Wang, and Yan Huang, 2009. Map-matching for low-sampling-rate GPS trajectories. In *Proceedings of the 17th ACM SIGSPATIAL international conference on advances in geographic information systems*. ACM, pp. 352-361.

[14] Hong Wei, Yin Wang, George Forman, Yanmin Zhu, and Haibing Guan, 2012. Fast Viterbi map matching with tunable weight functions. In *Proceedings of the 20th International Conference on Advances in Geographic Information Systems*. ACM, pp. 613-616.

[15] Edsger W. Dijkstra, 1959. A note on two problems in connexion with graphs. *Numerische mathematik*, *1* (1), pp. 269-271.

[16] Sebastian Knopp, Peter Sanders, Dominik Schultes, and Dorothea Wagner, 2007. Computing many-to-many shortest paths using highway hierarchies. In *Proceedings of the Ninth Workshop on Algorithm Engineering and Experiments (ALENEX)*. Society for Industrial and Applied Mathematics, pp. 36-45.

[17] Moritz Hilger, Ekkehard Köhler, Rolf H. Möhring, and Heiko Schilling, 2009. Fast point-to-point shortest path computations with arc-flags. In *The Shortest Path Problem: Ninth DIMACS Implementation Challenge*, American Mathematical Society, pp. 41-72.

[18] Robert Geisberger, Peter Sanders, Dominik Schultes, and Christian Vetter, 2012. Exact routing in large road networks using contraction hierarchies. *Transportation Science*, *46* (3), pp. 388-404.

[19] Hannah Bast, Daniel Delling, Andrew V. Goldberg, Matthias Müller-Hannemann, Thomas Pajor, Peter Sanders, Dorothea Wagner and Renato F. Werneck, 2016. Route planning in transportation networks. In *Algorithm Engineering*. Springer International Publishing, pp. 19-80.

[20] Daniel Delling , Andrew V. Goldberg and Renato F. Werneck, 2011. Faster batched shortest paths in road networks. *OASIcs-OpenAccess Series in Informatics*, *20*.

[21] Tetsuo Shibuya, 2000. Computing the nxm shortest path efficiently. *Journal of Experimental Algorithmics (JEA)*, *5* (9).

[22] Sebastian Knopp, 2006. *Efficient computation of many-to-many shortest paths*. Diplomarbeit, Universität Karlsruhe.

[23] Andrew V. Goldberg, and Chris Harrelson, 2005. Computing the shortest path: A search meets graph theory. In *Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*. pp. 156-165.

[24] Zhe Zeng , Tong Zhang, Haixiang Zou and Zhongheng Wu, 2015. Acceleration of Map Matching for Floating Car Data by Exploiting Travelling Velocity. In *IEEE 18th International Conference on Intelligent Transportation Systems (ITSC)* , pp. 2895-2899.

[25] Can Yang and Győző Gidófalvi, 2018. Fast map matching, an algorithm integrating hidden Markov model with precomputation. *International Journal of Geographical Information Science*, *32*, pp. 547-570.

[26] Ravindra K. Ahuja, Thomas L. Magnanti, James B. Orlin, 1993. *Network flows: theory, algorithms, and applications*. Prentice-Hall Inc.

[27] Mo Chen, Rezaul A. Chowdhury, Vijaya Ramachandran, David L. Roche, and Lingling Tong, 2007. *Priority queues and Dijkstra's algorithm*. Technical Report TR-07-54. University of Texas at Austin.

[28] Arvind Thiagarajan, Lenin Ravindranath, Katrina LaCurts, Samuel Madden, Hari Balakrishnan, Sivan Toledo and Jakob Eriksson, 2009. VTrack: accurate, energy-aware road traffic delay estimation using mobile phones. In *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems*. ACM, pp. 85-98.

[29] Chong Y. Goh, Justin Dauwels, Nikola Mitrovic, Muhammad T. Asif, Ali Oran, and Patrick Jaillet, 2012. Online map-matching based on hidden markov model for real-time traffic sensing applications. In *15th IEEE Intelligent Transportation Systems Conference (ITSC)*. pp. 776-781.

[30] Ming Xu, Yiman Du, Jianping Wu, and Yang Zhou, 2015. Map matching based on conditional random fields and route preference mining for uncertain trajectories. *Mathematical Problems in Engineering*, *2015*.

[31] Jian Yang and Liqiu Meng, 2015. Feature selection in conditional random fields for map matching of GPS trajectories. In *Progress in Location-Based Services 2014*. Springer International Publishing, pp. 121-135.

[32] George R. Jagadeesh and Thambipillai Srikanthan, 2017. Online Map-Matching of Noisy and Sparse Location Data With Hidden Markov and Route Choice Models. *IEEE Transactions on Intelligent Transportation Systems*, *18* (9), pp. 2423-2434.