Optimal Mobile Facility Localization

A.K.M. Mustafizur Rahman Khan
University of Melbourne
Victoria, Australia
khank@student.unimelb.edu.au

Lars Kulik
University of Melbourne
Victoria, Australia
lkulik@unimelb.edu.au

Egemen Tanin
University of Melbourne
Victoria, Australia
etanin@unimelb.edu.au

Tanzima Hashem
Bangladesh University of Engineering & Technology
Dhaka, Bangladesh
tanzimahashem@cse.buet.ac.bd

ABSTRACT

We introduce a new type of spatial query, Optimal Accessible Location (OAL) query. When a set of paths is provided the query finds the best location from a set of locations that has the optimal accessibility from these paths. OAL queries have many applications such as the selection of the optimal location for a mobile facility such as a food truck or selection of a venue for an event. We exploit geometric properties and develop pruning techniques to eliminate unrelated path segments as well as locations. Our experimental results demonstrate that we provide a readily deployable solution for real-life applications.

Categories and Subject Descriptors

H.2.8 [**Database Applications**]: Spatial databases and GIS

Keywords

Spatial database, facility localization, trip query processing

1. INTRODUCTION

Finding an optimal accessible location is necessary when a fixed location may not be convenient, for example, for mobile services such as a mobile food truck businesses. Therefore, dynamic selection of an optimal location considering on-the-fly customers is required. We introduce an important class of spatial query, an optimal accessible location (OAL) query. When a set of paths is provided the query finds the best location from a set of locations that has the optimal accessibility from these paths. In an OAL query, each path is represented as a set of line segments and the distance between a location and a path is the minimum distance from the location to any segment of the path. We refer to a location such as a car park as a point of interest (POI). To the best of our knowledge, we are the first to propose the OAL query and to develop an efficient approach for evaluating OAL queries.

A naive approach to solving the OAL query would first calculate the distances for a single POI to all segments of a path and then retrieve the minimum distance. Similarly, the distances for all paths from that POI are calculated, and afterwards the aggregate distance (AD) of that POI is obtained. Then the AD of each POI is computed, and the POI with the minimum AD is the desired POI. This brute force method incurs high computation costs.

SIGSPATIAL'15, November 03-06, 2015, Bellevue, WA, USA
©2015 ACM. ISBN 978-1-4503-3967-4/15/11...$15.00
DOI: http://dx.doi.org/10.1145/2820783.2820862

We exploit the fact that due to geometric constraints the OAL cannot be in some parts of the search space. To achieve this, we index the POIs with a tree-based data structure and map each path to this tree. We divide the total search space into smaller regions. We execute the query considering the POIs that falls inside each region and the nearby path segments. As a result, we do not build indices for segments that cannot provide a solution such as the regions without a POI or any nearby paths. In addition, geometric properties are also used to prune POIs within a region. We also estimate lower distance bounds of POIs from all paths to prune additional POIs.

2. RELATED WORK

The optimal facility location problem is to find out the best location for a facility from a set of locations. Max-inf optimal facility location problems maximize the "influence" of a facility, where influence is typically defined as the total weight of its reverse nearest neighbors (RNN). Cabello et al. [1] introduced the nearest location circle (NLC) to find the region with the maximum influence. Xia et al. [6] used a branch and bound method to find top-k influential facilities in a set of existing facilities within a continuous region Q. Zhang et al. [9] proposed the min-dist optimal-location problem. Given a set of existing sites, a set of weighted objects, and a spatial region Q, the query returns a location in Q which, if a new site is built there, minimizes the average distance from each object to its closest site. Xiao et al. [7] further studied the min-dist problem in road networks. Lin et al. [3] proposed a solution to find out the optimal site from a given set of sites S which has the maximum accessibility to amenities. The accessibility cost of a site to an amenity of a certain type is the shortest distance from that site to any amenity of that type. The GNN query finds the POI (meeting point) from a set of POIs that minimizes the aggregate distance to a group of users, located at stationary point locations [4]. Khan et al. considered privacy in GNN queries [2]. The Group Nearest Neighbor (GNN) query can be considered as a special case in an OAL query, where each path is just a point.

These studies consider objects located in points. The objects in our problem are complex paths and each path is a set of line segments. Therefore, the above mentioned solutions are not applicable and novel techniques to solve the OAL query is required.

Given a set of paths, Shang et al. [5] considered finding the facility that is the nearest facility to the maximum number of paths. We search for the location that has the minimum aggregate distance to the given paths. Thus the function to be optimized in [5] is significantly different than ours. They pre-process the set of paths to efficiently process a query assuming that the paths are available beforehand and a list of candidate locations is provided as query input. We consider that the data of candidate locations are available beforehand and the set of paths are received as query input.

3. OPTIMAL ACCESSIBLE LOCATIONS

3.1 Problem Definition

Let $T = \{t_1, t_2, \ldots, t_n\}$ be a set of paths and P be the set of POIs. A path t_i is described as $E_i = \{e_{i_1}, e_{i_2}, \ldots\}$, where E_i is the set of edges/segments of the path. Let e_{ij} be the closest segment of all segments of t_i to a point p. The distance from p to path t_i is defined as $d(p,t_i) = dist(p,e_{ij})$. Here $dist(p,e_{ij})$ represents the Euclidean distance of e_{ij} from p. The amount of diversion if a person travelling on t_i goes to p is $d(p,t_i)$. The OAL query finds a data point q from P, such that for any $q' \in P - \{q\}$, $\sum_{i=1}^{n} d(q,t_i) \le \sum_{i=1}^{n} d(q',t_i)$. We call q the Optimal Accessible Location (OAL). The customers provide their paths $\{t_1, t_2, \ldots, t_n\}$ as query data and the location service provider (LSP) returns the OAL q.

3.2 A Naive Approach

A naive approach considers all combinations of segments of all paths and POIs. The distance of each path from each POI is calculated. To calculate distance between a path and a POI, one calculates and compares the distances of all segments of that path to the POI. Then for each POI the aggregate distance (AD) to all paths is calculated. The POI with the smallest AD is declared as the OAL.

A naive algorithm requires distance calculations for each combination of POI and segment. Let $|P|$, $|T|$ and $|E|$ be the number of POIs, number of paths and the number of segments per path. Hence, the computational complexity is $|P||T||E|$. For large values of $|P|$, $|T|$ and $|E|$, processing an OAL query by a brute force algorithm incurs significantly high computational cost. To speed-up this approach we can use a simple "early stop" pruning method. We prune a POI if the summation of the distances from the POI to some paths are larger than the smallest AD seen so far. In this case, we avoid calculating distance of the remaining paths from that POI.

3.3 Lazy Divide and Prune (LDP) Approach

We assume that the POIs are indexed with a disjoint space partitioning based data structure. Hence the whole search space, which includes all POIs, is partitioned into multiple cells. After receiving the paths as query data, we map them onto the partitioned search space to locate precisely the regions that have a high probability of containing the OAL. We search the cells for the OAL one by one considering their probability of including the OAL. We use multiple geometric constraints and a novel bounded search algorithm to prune the POIs while searching a cell. The POI with the minimum AD is declared as the OAL after searching all cells.

3.3.1 Indexing POIs

We assume that the POIs are indexed with a disjoint space partitioning based tree structure and a leaf contains at most one POI. We use a quadtree because of its simplicity. We call the quadtree built on POIs a "P-tree". We refer to a node of a tree by its corresponding space/cell. If a cell contains more than one POI, the cell is divided into four equal square sized non-overlapping cells. An example of a P-tree is shown in Figure 1(a), where asterisk marks represents POIs. The horizontal and vertical lines (both solid and dashed) define the space partitioning of the P-tree.

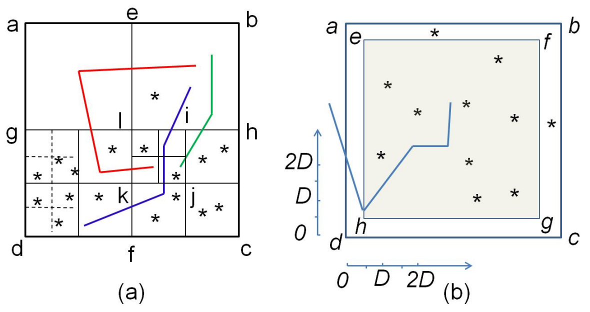

Figure 1: Examples of (a) Mapping segments on a P-tree and (b) Pruning POIs considering distance from a block's boundary.

3.3.2 Mapping Paths and Dividing Search Space

After receiving the query data, which consists of multiple paths, we map the paths to the P-tree built on the POIs in the indexing step. The mapping helps to precisely locate the cells of the P-tree that have a high probability of containing the OAL. A dense cell, which contains many paths and POIs, has a high likelihood to include the OAL. We start searching from the cells that have high probability. In addition, the mapping facilitates the division of the problem to smaller sub-problems. Each sub-problem considers the POIs in a cell and the surrounding line segments only. Moreover, POIs and line segments can be pruned considering their distances from the boundary of a cell.

We map the paths from the root to the leaves of the P-tree. If a cell corresponding to a node contains POIs and many paths pass through it, it has a high likelihood to include the OAL. If a cell is intersected by more than α paths, we further map the paths on the children cells. We call α a division threshold. If a cell is not intersected by enough paths or does not include a POI we stop mapping to the children of the cell. In this case, if the cell contains at least one POI, we call the cell a q-block. If the cell is intersected by any path, we call it a s-block. The mapping stops when there is no cell remaining that is intersected by at least two paths and contains more than one POI. A list, Q, is maintained to record the q-blocks' information. The OAL will be searched in the recorded q-blocks. Another list S is used to record s-blocks. The s-blocks collectively contain all segments of all paths. Note a block must be a cell but not all cells are blocks. A cell represents the space corresponding to a node/leaf of a tree. A block is a cell where mapping of segments terminates.

We illustrate the path mapping procedure with an example depicted in Figure 1(a). Three paths (i.e., red green and blue lines) in the figure are mapped on the P-tree built on POIs. In this example, if at least two paths (i.e., $\alpha = 2$) intersect a cell which contains more than one POI, the paths are mapped to children cells. $abcd$ represents the root of the P-tree and the paths are mapped on children cells, i.e., $aelg$, $ebhl$, $lhcf$ and $glfd$. Since $aelg$ and $ebhl$ do not contain enough POIs, the path mapping stops in these cells. $aelg$ and $ebhl$ are registered as s-blocks as they are intersected by segments. $ebhl$ is listed as a q-block as it contains a POI. Since both $lhcf$ and $glfd$ are intersected by at least two paths and contains many POIs, the paths are mapped to their children cells. Among the children cells of $lhcf$ and $glfd$ only $ijkl$ is selected for further mapping. In this way the paths are mapped on the P-tree. The mapping stops when there is no cell remaining that is intersected by at least two paths and contains more than one POI. Thus none of the children cells of $ijkl$ is selected for further mapping and the path mapping procedure terminates. The solid horizontal and vertical lines in Figure 1(a) define the final space partitioning of the search space. Each part with a solid line boundary represents a block.

3.3.3 Query Simplification and Pruning

We simplify the query by dividing it to smaller sub-problems. In each sub-problem we consider a q-block registered in Q, and check whether the q-block contains the OAL. We call the nearest point of the OAL on a path a *striking point*. The *base block* of a POI is the block which contains the POI. Let at a certain moment during query processing the POI with minimum AD be *CurrentBest* and the current minimum AD be AD_{Cmin}. When considering a q-block we apply the following pruning techniques.

Pruning POIs Using Distance From the Boundary of a Block: If a block is not intersected by all paths and it contains the OAL, the OAL must be close to the boundary of the block. Suppose a block contains α' paths out of $|T|$ paths, where $\alpha' < |T|$. Hence, the block cannot contain more than α' striking points and at least $|T| - \alpha'$ striking points are outside of it. If the OAL is inside the block and x is the distance of the OAL to the boundary of the block, the AD of the striking points outside the block to the OAL is at least $x(|T| - \alpha')$. Since $x(|T| - \alpha') < AD_{Cmin}$, the distance of the OAL to the boundary of the block cannot be larger than $\frac{AD_{Cmin}}{|T|-\alpha'}$. If the distance of a POI to the boundary of its base block is larger than $\frac{AD_{Cmin}}{|T|-\alpha'}$, it can be pruned.

In Figure 1(b), block *abcd* is intersected by only one path out of three paths and D is the current minimum AD. Hence, the maximum distance of the OAL from the boundary of *abcd* can be $x/(3-1) = D/2$. Let $efgh$ be a square inside *abcd* and its distance from the boundary of *abcd* be $D/2$. Hence, the OAL cannot be in $efgh$ and all POIs inside $efgh$ can be pruned.

Pruning with Lower Bound on AD: While considering a POI p as a candidate for OAL, we incrementally search for possible striking points and estimate the lower bound on AD of p. If the lower bound of AD of a POI is larger than the current minimum AD, the POI cannot be the OAL. The process is similar to nearest neighbor search in a tree. We incrementally search the neighborhood for the nearest segment of each path to a POI and estimate the lower bound on distances of the unfound paths using the radius of the already searched space. First, we calculate the minimum distances of the paths considering only the segments that intersects the base block. The base block of p is the block which contains it. After considering all segments of the base block, we consider neighboring s-blocks in an increasing order of distances from the base block.

Let the base block of a POI p and all blocks within x' distance of the base block represent the already searched space. Let the distance from p to the nearest segment of path t_i that intersects the searched space be $d_{Cmin}(p,t_i)$. If the distance of the boundary of the base block is x from p, the distances of the segments and paths that do not intersect the searched space from p must be at least $x+x'$. Hence, the lower bound of the distance of path t_i from p is $\min\{d_{Cmin}(p,t_i), x+x'\}$. If $d_{Cmin}(p,t_i) \leq x+x'$, the distance of path t_i from p is $d_{Cmin}(p,t_i)$, since the distance of the segments that do not intersect the searched space is at least $x+x'$.

Suppose α' paths intersect the searched space and the set of these paths is T_f. So the lower bound on AD of the paths that do not intersect the searched space is $AD_{LB}(p, T - T_f) = (|T| - \alpha')(x + x')$. The lower bound of AD of the paths in T_f is

$$AD_{LB}(p,T_f) = \sum_{t_i \in T_f} \min\{d_{Cmin}(p,t_i), x+x'\} \tag{1}$$

Hence the lower bound of AD of p,

$$AD_{LB}(p,T) = (|T| - \alpha')(x+x') + \sum_{t_i \in T_f} \min\{d_{Cmin}(p,t_i), x+x'\}. \tag{2}$$

If $AD_{LB}(p,T) > AD_{Cmin}$, p cannot have an AD lower than that of the current best POI *CurrentBest* and thus cannot be the OAL. If for all paths $x+x' \geq d_{Cmin}(p,t_i)$, the AD of p is

$$AD(p,T) = \sum d_{Cmin}(p,t_i) \tag{3}$$

3.3.4 Reducing Indexing Cost With a Lazy Tree

If the number of segments of a certain path inside a s-block is large, we employ an efficient method to find the nearest segment to a POI. Our method simultaneously index the segments and search for the nearest segment of a POI. We index the segments of each intersecting path of a block with a tree in a lazy fashion. We call the tree a *lazy tree*. A separate lazy tree is constructed for each intersecting path of each s-block. We find the nearest segment of a path to a POI by searching the lazy tree. The lazy tree is updated during a search by indexing line segments to a deeper level if necessary. During a nearest segment search, first we find the closest node of the lazy tree to the concerned POI. Let the maximum allowable number of segments in a leaf of a lazy tree be β. If more than *beta* segments are attached to it, we create children nodes by dividing the node's space into four equal non overlapping square sized cells. Then we distribute the segments to children nodes. Next we find the nearest cell to the concerned POI. We refer to this cell as a *base cell*. We find the nearest segment to the POI inside the base cell. If there is a neighboring cell closer than the current nearest segment to the POI, we consider that neighboring cell. We split a neighboring cell of a base cell if it satisfies the following two conditions:

1. The neighboring cell is intersected by more than β segments of the concerned path.

2. The neighboring cell has not been created during this search.

3.3.5 Searching the OAL

We sort the q-blocks according to the number of intersecting paths. Next we sort the q-blocks with equal number of intersecting paths in an increasing order of size. We initialize the current minimum AD AD_{Cmin} with a large value and *CurrentBest* with a randomly selected POI. Then we consider the q-blocks one by one. Let α' paths for total T paths intersect a q-block. As mentioned in Section 3.3.3, we prune the POIs whose distance from the boundary of the q-block is larger than $\frac{AD_{Cmin}}{|T|-\alpha'}$. If there is any un-pruned POI, we search for the nearest segment of each path to the POI and estimates the lower bound on AD of the POI. First, we consider the segments inside the base block. Next, we search the neighbouring s-blocks. Let the distance of the boundary of the base block from the POI p is x and x' is the distance of the farthest neighbouring s-block from the base block considered so far. While considering a POI p as a candidate, we first search the base block ($x' = 0$) and then keep on increasing x' until the lower bound of AD of the POI exceeds AD_{Cmin} or for all paths $x+x' \geq d_{Cmin}(p,t_i)$. Suppose α' paths intersect the already searched space and the set of these paths is T_f. We calculate the lower bound on AD of the POI using Equation (2). If the lower bound on AD of p is larger than AD_{Cmin}, p is pruned. If for all paths $x+x' \geq d_{Cmin}(p,t_i)$, the AD of p is $AD(p,T) = \sum d_{Cmin}(p,t_i)$. If $AD(p,T) < AD_{Cmin}$, AD_{Cmin} and *CurrentBest* are updated to $AD(p,T)$ and p. After considering all q-blocks, the *CurrentBest* is declared as the OAL.

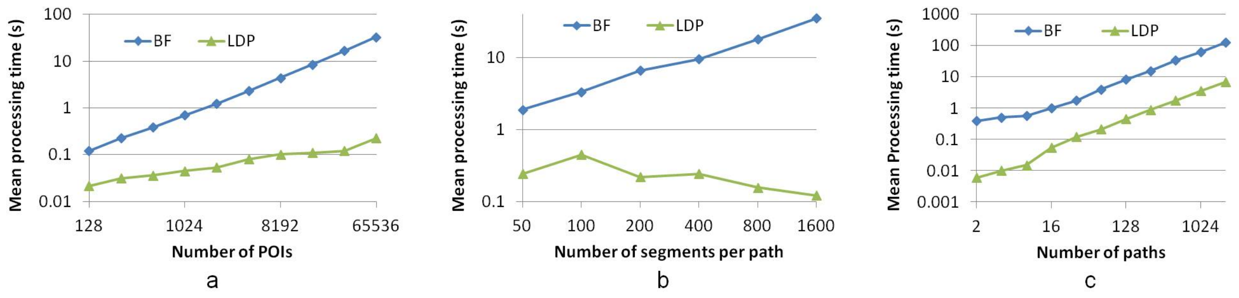

Figure 2: Effect of (a) number of POIs, (b) length of path and (c) number of paths on the processing time

4. EXPERIMENTAL EVALUATION

4.1 Experimental Set-up

In our experiments we used the T-drive data[8] which contains 30 days of GPS trajectories of 6400 taxis in Beijing. We controlled a range of parameters including the number of paths, the number of POIs, the number of segments per path. We randomly selected taxis and used their trajectories as paths. We extracted 531 locations of restaurant in Beijing from OpenStreetMap. We generated artificial restaurant locations by adding Gaussian noise to the extracted locations. We considered 100 queries for each set of experiments, evaluated the proposed algorithms for each of these queries and determined the average results. For each query we generated a new set of POIs' locations and selected a new set of paths. We set the division threshold to the half of the number of paths (i.e., $\alpha = |T|/2$). We ran our experiments on a desktop computer with an Intel Core i7-2600 3.40GHz processor and 8GB RAM.

4.2 Results and Discussion

We refer to the naive method and the lazy divide and prune method as BF and LDP respectively. Effect of various parameters are discussed below.

4.2.1 Number of POIs

To estimate the scalability of the algorithms to the number of POIs, we ran experiments varying the number of POIs within the range of 128-65536. The number of paths were 64 and number of segments per path was 200. The query processing time of each algorithm is presented in Figure 2(a). Figure 2(a) shows that LDP is about two to three orders of magnitude faster than BF in all cases. When there are 65536 POIs LDP takes only 0.25s indicating that it can be practically useful. If several hundred queries, which can be generated in a big city, are issued simultaneously, BF will be unable to response in a reasonable time period.

4.2.2 Path Length

We varied the number of segments from 50 up to 1600. The numbers of POIs and paths were 16384 and 64 respectively. The average length of the paths increased linearly from 18Km to 520Km with the increase of number of segments from 50 to 1600. The processing time of different algorithms are shown in Figure 2(b). LDP is two to three orders of magnitude faster than baseline methods. If the paths are longer, it is more likely that a better meeting place can be found. As result, for longer paths, the AD of the OAL tends to be smaller. Since pruning efficiency depends on the AD_{Cmin}, it directly effects the processing time. Thus the processing time decreases with the increase of path length for LDP.

4.2.3 Number of Paths

We varied the number of paths from 2 to 2048. The numbers of POIs and segments per path were 16384 and 200 respectively. Figure 2(c) shows that, LDP outperforms the baseline algorithm. LDP is one to two order of magnitude faster than baseline method.

5. CONCLUSION

We have introduced an important new query, the OAL query, and provided an efficient solution for finding the best accessible location for a group of paths. We have compared our solution against a baseline technique that includes an early stop pruning condition to make it more competitive. Although our solution also uses a tree, we instead map all paths to a single tree and prune all path segments that cannot be part of the optimal solution by exploiting relative positioning of the paths and thus simplify the query. We also develop an efficient technique to estimate lower distance bounds of POIs from all paths to prune POIs. Our experimental result shows that our methods outperforms the baseline method by two to three orders of magnitude.

6. REFERENCES

[1] S. Cabello, J. M. Díaz-Báñez, S. Langerman, C. Seara, and I. Ventura. *Reverse facility location problems*. University of Ljubljana, Inst. of Mathematics, Physics and Mechanics, Department of Mathematics, 2006.

[2] A. M. R. Khan, T. Hashem, E. Tanin, and L. Kulik. Location oblivious privacy protection for group nearest neighbor queries. In *Geographic Information Science*, pages 301–317. 2014.

[3] Q. Lin, C. Xiao, M. A. Cheema, and W. Wang. Finding the sites with best accessibilities to amenities. In *DASFAA*, pages 58–72, 2011.

[4] D. Papadias, Q. Shen, Y. Tao, and K. Mouratidis. Group nearest neighbor queries. In *ICDE*, pages 301–312, 2004.

[5] S. Shang, B. Yuan, K. Deng, K. Xie, and X. Zhou. Finding the most accessible locations: reverse path nearest neighbor query in road networks. In *SIGSPATIAL*, pages 181–190, 2011.

[6] T. Xia, D. Zhang, E. Kanoulas, and Y. Du. On computing top-t most influential spatial sites. In *VLDB*, pages 946–957, 2005.

[7] X. Xiao, B. Yao, and F. Li. Optimal location queries in road network databases. In *ICDE*, pages 804–815, 2011.

[8] J. Yuan, Y. Zheng, C. Zhang, W. Xie, X. Xie, G. Sun, and Y. Huang. T-drive: driving directions based on taxi trajectories. In *SIGSPATIAL*, pages 99–108, 2010.

[9] D. Zhang, Y. Du, T. Xia, and Y. Tao. Progressive computation of the min-dist optimal-location query. In *VLDB*, pages 643–654, 2006.