Continuous Detour Queries in Indoor Venues

Chaluka Salgado
chaluka.salgado@monash.edu
Monash University
Clayton, Australia

Muhammad Aamir Cheema
aamir.cheema@monash.edu
Monash University
Clayton, Australia

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

ABSTRACT

In this paper, we study continuous detour queries in the indoor space. A continuous detour query finds the nearest indoor detour object like an ATM or a printer for a moving user walking towards a target location in an indoor venue, where the detour distance for an indoor object is measured as the total indoor distance of the object from the user's current and target locations. The continuous detour query has been already studied for the outdoor space, but the solutions are not adaptable for the indoor space due to the unique characteristics of indoor venues. We develop the first solution for efficient processing of the continuous detour query in the indoor space. The novelty of our solution comes from the computation of safe zones for the indoor objects by exploiting the geometric properties of hyperbolas, additively weighted Voronoi diagram and indoor partitions. The safe zone represents an area such that the nearest detour object remains unchanged as long as the user is in this area. The key ideas behind the efficiency of our solution are reducing the number of re-evaluation of the detour queries for the location change of a moving user, pre-computing the safe zones, and indexing them using a grid structure. The experiments show that our solution can process continuous detour queries efficiently and reduces the communication overhead.

CCS CONCEPTS

• **Information systems → Location based services**.

KEYWORDS

detour query, indoor , safe zone, continuous query, client-server

ACM Reference Format:
Chaluka Salgado, Muhammad Aamir Cheema, and Tanzima Hashem. 2019. Continuous Detour Queries in Indoor Venues. In *16th International Symposium on Spatial and Temporal Databases (SSTD '19), August 19–21, 2019, Vienna, Austria.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3340964.3340977

1 INTRODUCTION

Indoor location-based services (LBSs) play an important role in planning daily activities with convenience as people usually spend most of their time inside the indoor venues like offices, shopping centers, transport facilities, and libraries. Although the efficient processing of various outdoor LBSs has been extensively studied over the last couple of decades, the indoor LBSs had not received significant attention due to various reasons [3]. Thanks to the recent advances in indoor positioning technologies, the indoor LBSs have started to receive attention from the research community. Recently, it was shown [17] that the outdoor query processing techniques [1, 2, 8, 29, 30] are not efficiently applicable for indoor LBSs due to the unique characteristics of indoor venues. For example, in the road network setting, people move along the roads whereas the buildings are normally organized into partitions with walls, doors, stairs and lifts, and people are free to move inside a partition. Thus, to continue the proliferation of indoor LBSs, it is essential to develop efficient solutions for processing location based queries in indoor venues.

In this paper, we propose the first solution for continuous detour queries in indoor venues, an essential class of LBSs that allows a moving user to continuously monitor the nearest detour object (e.g., an ATM, a printer, check-out counter) while walking towards a target location in an indoor venue. For example, a user in a library may want to visit a check-out kiosk before returning to the car park. A detour query returns the indoor object that has the smallest detour distance, where the detour distance is measured as the total indoor distance of an indoor object from the user's current and target locations. However, it is common that an indoor user deviates from her planned path to the target location which may result in a change to the nearest detour object. For example, the library user may deviate from her current path to have a look at some books along her way or a user in a shopping centre may enter a store to look at some products. Furthermore, the users may also miss a turn while walking towards the nearest indoor detour object. In such scenarios, the nearest indoor detour object may continuously change as the user walks which requires continuously updating the results.

Processing a continuous detour query using a detour query for every location update of a moving user would incur a high processing overhead. Thus, the efficiency of a continuous detour query processing algorithm for indoor venues highly depends on reducing the number of re-evaluations of the detour queries. This also results in excessive communication cost between the client and the server because the up-to-date results need to be sent to the user at each timestamp. To address this issue, in addition to the current nearest indoor detour object, our solution provides a user with additional information so that the user does not need to communicate with the server for the re-evaluation of the detour query for moving within a specific area.

SSTD '19, August 19–21, 2019, Vienna, Austria
© 2019 Association for Computing Machinery.
ACM ISBN 978-1-4503-6280-1/19/08…$15.00
https://doi.org/10.1145/3340964.3340977

Since the density of the indoor objects is much higher than that of the outdoor space, performing all computations in query time is not feasible for the indoor space. We develop a novel technique to compute the *safe zones* for indoor objects by exploiting the geometric properties of hyperbolas, additively weighted Voronoi diagrams [12] and indoor partitions. A safe zone for an indoor object represents an area where it is guaranteed that the indoor object remains the nearest detour object with respect to a partition door for a user who is moving towards a fixed target point. Our safe zone computation technique does not depend on any query time parameter and thus, allows us to pre-compute the safe zones and use them for query processing. Pre-computed safe zones significantly reduce the query computational overhead for identifying the nearest detour objects, and the communication overhead by allowing the user to have the nearest detour objects for a guaranteed area without sending a re-evaluation request of the detour query to the location-based service provider.

The underlying idea of our solution is to divide the problem of finding a nearest detour for a user's current and target locations into two subproblems: identifying the *minimum local detour* and the *minimum remote detour* with respect to an indoor partition. For the minimum local detour, using the stored safe zones, we find the indoor object that has the smallest detour distance among all indoor objects in the indoor partition where the user is currently located. We also develop an efficient algorithm to identify the indoor object that has the smallest detour distance among all indoor objects outside the user's current partition and we consider the identified indoor object as the minimum remote detour. Finally, the indoor object that provides the smallest detour distance between the minimum local detour and the minimum remote detour is selected as the query answer.

The query answer, a grid cell representing an area that may overlap with the safe zones of one or multiple indoor objects along with other additional information are sent to the user. The query answer may change as the user moves but as long as the user resides within the grid cell (denoted as the client-side safe zone), the user can compute the new nearest indoor detour object without communicating with the server. Moreover, the user's location is sent to the server for finding the nearest indoor detour object, if the user moves outside the client-side safe zone.

Continuous detour queries have been addressed in the outdoor space [11, 13]. In the road networks, users are restricted to move towards the roads and thus, the existing solution [11] for processing continuous detour queries in the road network is not applicable for indoor scenarios. On the other hand, [13] continuously monitors the nearest detour objects in the outdoor space in presence of obstacles like a building, a river or a fence. Due to the random distribution of the obstacles in the outdoor space, the solution [13] performs all computations during query time and is prohibitive for the indoor space.

The contributions of this paper are summarized as follows:

- We are the first to study the continuous detour queries in the indoor space.
- We propose an efficient solution for continuously monitoring the nearest indoor detour object for a user walking towards a fixed target location.
- We develop a novel technique to pre-compute the safe zones for indoor objects inside a partition.
- We perform an extensive set of experiments and show that our solution performs significantly better than the competitors.

The remainder of the paper is organized as follows. In Section 2, we discuss the works related to our problem. In Section 3, we formulate the problem of the continuous detour query. Section 4 presents our solution for efficient processing of the continuous detour queries. In Section 5, we present the experimental results and in 6, we conclude the paper with the future research direction.

2 RELATED WORK

2.1 Query Processing in the Indoor Space

The literature [17] shows that the existing outdoor query processing techniques fall short in indoor space as they do not consider unique properties of an indoor space such as hallways and rooms. Thus, the efficient indoor query processing has received a significant attention recently. A large body of work for querying indoor data, shortest distance/path, range and k nearest neighbour queries under various settings can be found in [10, 21, 22, 27].

The indexing structures that take into account the uniqueness of the indoor space have been proposed for efficient indoor query processing. Xie et al. [20] develop a composite indexing structure called *ind*R-tree, that indexes indoor entities into different layers, namely the geometric layer, the topological layer, and the object layer. Yang et al. [23] propose a distance-aware indoor space model called accessibility base (AB) graph that describes the topology of a indoor space by capturing the fundamental connectivity and accessibility.

D2D graph [23] is one of the most notable techniques which has been used in most of the studies in literature since they enable various query processing techniques in road networks [2, 29, 30] to be applied in the indoor space. D2D graph represents doors in the indoor space as vertices. A weighted edge between two vertices is created if they are connected to the same indoor partition (e.g., room, hallway) where the edge weight is the indoor distance between the corresponding doors. Lu et al. [9] propose a distance aware indoor space data model along with efficient distance computation algorithms. To support efficient distance aware queries, they use a distance matrix, which is derived from the indoor topology and indoor distances. Also, they propose a room-based expansion algorithm to process distance queries along with accessibility graph which captures the essential connectivity and the accessibility of the indoor space.

Shao et al. [17] introduce an efficient indexing structure called IP-tree that takes into account unique indoor properties in tree construction and query processing. In an IP-tree, adjacent indoor partitions (e.g., rooms, hallways, staircases) are combined to form leaf nodes. Then, the adjacent leaf nodes are combined to form intermediate nodes. This process is iteratively continued until all nodes are combined into a single node (i.e., root node). VIP-tree [17] is an improvement of the IP-tree. Compared to the existing indexing techniques, VIP-tree has demonstrated more efficiency and higher scalability. Thus, we use the VIP-tree as our indexing structure in this study.

2.2 Detour Queries in the Outdoor Space

In-Route Nearest Neighbor Queries (IRNN) in [18, 26] are designed assuming users are stick to a fixed daily route where they even return back to the particular route after visiting a detour facility. Hence, they compute the nearest neighbours with the minimum detour distance from the fixed route. Chen et al. [7] introduce Path Nearest Neighbour (PNN) which capable of monitoring nearest neighbour for both scenarios where user move along the preferred path and user deviated from the preferred path. Basically, they monitor the nearest neighbour for a dynamically changing path rather than a query point. Hence, these works cannot be extend to answer detour queries. Shang et al. [15] study a detour problem where a preferred path is given along with a detour distance threshold. They initially divide the path into set of segments and find candidates whose detour distances less than the given threshold. Then they find the best detour point after validating the candidates. Thus, these techniques cannot be extended as the path must be predetermined.

[11] proposes a solution where an order-k shortest path tree is incrementally build. In an indoor graph, door to door edge represent an indoor partition which may consists of many objects. Hence, constructing a shortest path tree is not possible for indoor space as the space inside the partition must be considered as Euclidean space. Even though they study a similar problem to ours, we find that their techniques cannot be extended to answer detour queries in indoor space. Moreover, [13] investigates detour queries in obstructed space. They propose an approach that constructs safe zones by retrieving $k + x$ nearest obstructed detour objects with respect to the user location. We find their work have different aims compared to our problem.

A variant of nearest neighbor queries called aggregated nearest neighbor can also be utilized to answer detour snapshot queries. Yiu et al. [25] propose three techniques that utilize Euclidean distance bounds, spatial access methods and network distance materialization structures. [19, 31] present approaches based on network voronoi diagram. Each of these approaches consist of two phases, namely, searching phase and pruning phase. In searching phase, they continuously search for next NN from each query point until a common object is found. After a candidate set is obtained, they continuously expand the search by computing the next NN for a certain query point while pruning unqualified objects. Zhang et al. [28] study a variant of ANN query that takes into account both spatial proximity and textual similarity. They propose an indexing schema called dual-granularity where bitmaps are integrated into a G-tree. They consider POIs on the edges as newly added vertices to the network graph. Thus, it may expand the scale of the original graph by several times, thereby leading to high space and query overheads. Briefly, we find all these techniques are inapplicable in answering detour queries in indoor space. Next, we discuss route planning queries in indoor space which can be used to answer snapshot detour queries. Shao et al. [16] are the first to study the indoor trip planning queries. They propose an exact solution called VIP-tree neighbour expansion (VNE). In this study, they assume that the number of indoor points in the indoor venues are very small. Thus, their techniques cannot be extended to answer detour queries. In [14], they get the aggregated distance to an indoor object from given two indoor points by utilizing VIP-tree indexing structure. We adopt their techniques in this study. Even though their techniques can be extended to answer detour snapshot queries, the performance is similar to the naive approach as they have to compute results per location update.

3 PROBLEM DEFINITION

D2D Graph. An indoor space consists of partitions and doors. A D2D graph [23] is used to represent an indoor space. In a D2D graph, each door is represented as a graph node. A weighted edge between two nodes is created if they are connected to the same indoor partition (e.g., room, hallway) where weight of the edge is the indoor distance between the corresponding doors. In this paper, we use D2D graph to model the indoor space because it is the one of the popular models and allows efficient query processing as shown in [17].

VIP-Tree. Shao et al. [17] introduce an indexing structure called IP-tree which is based on a D2D graph. In an IP-tree, adjacent indoor partitions (e.g. rooms, hallways, staircases) are combined to form leaf nodes. Then, the adjacent leaf nodes are combined to form intermediate nodes. This process is iteratively continued until all nodes are combined into a single node (i.e., root node). VIP-Tree or Vivid IP-Tree is an improvement of the IP-tree. Further details can be found in [17] . We utilize the VIP-tree to index the indoor space.

Indoor Objects. Let $p_i \in \mathcal{P}$ be an indoor point representing an indoor object[1]. The location of an indoor point p_i is given by x and y coordinates.

Definition 3.1 (Detour). Given a set of indoor points $\mathcal{P}$, the detour $D = \{p_s, p_i, p_t\}$ is a route from a given source indoor point p_s to a given target indoor point p_t going through an indoor point $p_i \in \mathcal{P}$ (denoted as detour point).

Definition 3.2 (Candidate Partition). Given a set of indoor points $\mathcal{P}$, an indoor partition I is called a candidate partition only if it consists of at least one indoor point $p \in \mathcal{P}$.

Definition 3.3 (Local/Remote Detour). Given a detour $D = \{p_s, p_i, p_t\}$ is called a **local** detour (denoted by D^L) if the detour point, i.e., p_i, belongs to the current indoor partition I where the user resides in. Otherwise, if the detour point lies outside the current indoor partition, it is called a **remote** detour (denoted by D^R).

Moreover, when the context is ambiguous, we define a local detour by $D^L_{i:j} = \{p_s \rightarrow p_i \rightarrow d_j \rightarrow p_t\}$ where p_i is an indoor point and d_j is a partition door. And a remote detour starts at door d_j by D^R_j.

Definition 3.4 (Detour Query). Given a set of indoor points $\mathcal{P}$, a query $q = \langle p_s, p_t \rangle$ where p_s denotes the current user location and p_t denote the target indoor point. The detour query returns the detour $D = \{p_s, p_i, p_t\}$ subject to:

$$L(D) = \underset{\forall p_i \in \mathcal{P}}{\textbf{arg min}}\ d(p_s, p_i) + d(p_i, p_t) \tag{1}$$

where $d(p_i, p_j)$ is the shortest indoor distance between indoor points p_i and p_j and the $L(D)$ is the indoor distance of the detour.

[1]In this paper, we use the terms indoor object and indoor point interchangeably.

Table 1: The summary of notations

Notation	Definition
p_i	an indoor point
q_i	a detour query
I	an indoor partition
$\mathcal{S}_{i:j}$	the splitter between points p_i and p_j
$H_{i:j}$	the region where $D^L_{i:j}$ is the minimum
$D^L_{i:j}$	the minimum local detour from point p_i via door d_j
D^R_i	the minimum remote detour at door d_i
$d(p_i, p_j)$	the indoor distance between points p_i and p_j
$L(D)$	the indoor distance of detour D

Definition 3.5 (Continuous Detour Query). Given an indoor space and a moving detour query where p_s continuously changes as a user moves. The continuous detour query continuously finds the minimum detour w.r.t the current user point p_s.

4 OUR SOLUTION

In this section, we develop an efficient solution for processing continuous detour queries in indoor venues. The key idea behind the efficiency of our solution is to use the concept of safe zones [4–6]. A safe zone is an area where the query results does not change as long as the query is inside it, i.e., the areas where indoor objects remain as the nearest detour objects for a user moving towards a fixed target location. We exploit the geometric properties of hyperbolas [24], additively weighted (AW) Voronoi diagrams (or AWVDs) and indoor partitions to compute such safe zones. A significant advantage of our safe zone computation technique is that it does not depend on the query time parameters and thus, allows us to pre-compute the safe zones and index them for use during the query evaluation. Since the density of indoor objects is much higher than the outdoor space, computing the safe zones during the query evaluation is not feasible for the indoor space. Pre-processed safe zones significantly reduce the computational complexity of finding the nearest detour objects and allow us to provide users with guaranteed areas where the users can locally determine the nearest detour objects without re-evaluating the detour queries for their changed locations. Since a user can freely move inside an indoor partition, the straightforward evaluation of a continuous detour query by using a detour query for every location update of the moving user would incur excessive processing overhead.

Our solution is based on the client-server paradigm for processing the continuous detour queries in the indoor space. The server is the location-based service provider and it is responsible for evaluating the continuous detour queries while the client is the user who issues a continuous detour query. The clients send location updates to the server as they move outside the guaranteed area. The server is responsible for maintaining the up-to-date results for the queries with respect to the user's movements. We divide the problem of finding the nearest detour objects into two subproblems, where we compute the minimum local and remote detours separately for each location update. Hence, the query result, i.e., the minimum detour, for a location update can efficiently be determined by utilizing the results of the subproblems. Intuitively, when a movement of a user does not invalidate the previous results of a subproblem, such results can be reused to obtain the minimum detour for the new location of the user. Section 4.1 presents the proposed techniques to pre-compute the safe zones, and find the minimum local detour with respect to a user's current and target locations. Section 4.2 introduces an efficient algorithm that utilizes the VIP-tree to find the minimum remote detour for a given door of an indoor partition. Section 4.3 explains the algorithm for evaluating a detour query and Section 4.4 presents the techniques for monitoring the nearest detour against the continuous location updates. Finally, Section 4.5 discusses a competitive algorithm that performs the *local computation* for a detour query by generating all possible local detours.

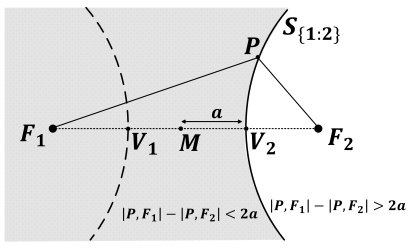


Figure 1: Example of a splitter

4.1 Local Computation

In this section, we explain the process of obtaining the minimum local detour for a given user location. We denote this process as the "local computation". We propose an efficient solution that utilizes safe zones to quickly identify the minimum local detour for a given indoor point. First, we introduce a pre-processing method to construct such safe zones. Then, we present an indexing method to store the pre-processing results, i.e., safe zones, to be utilized in query processing.

4.1.1 Pre-processing. Naively, the minimum local detour can be determined by generating all possible local detours. Since the indoor space consists of thousands of objects, such an approach is computationally expensive. By exploiting the geometric properties of the hyperbolas, we identify safe zones in the indoor space, where a user's movement does not change the current local detour point that provides the smallest detour distance with respect to a door of an indoor partition. Since these safe zones are independent of the query parameters, we pre-compute and utilize them in the query processing to perform the local computation efficiently. Before we present the pre-processing method, we introduce the following definitions.

As Figure1 shows, a hyperbola is a set of points, such that for any point P of the set, the absolute difference of the distances to two fixed points F_1, F_2 (the foci), is constant, i.e. , $H = \{P \mid |d(P, F_1) - d(P, F_2)| = 2a\}$. Note that the curve goes through vertex V_2 divides the space into two half spaces where $d(\hat{P}, F_1) - d(\hat{P}, F_2) < 2a$ if the point $\hat{P}$ in the left half space and $d(\hat{P}, F_1) - d(\hat{P}, F_2) > 2a$ if the point $\hat{P}$ in the right half space. Hence, we define such a curve as follows.

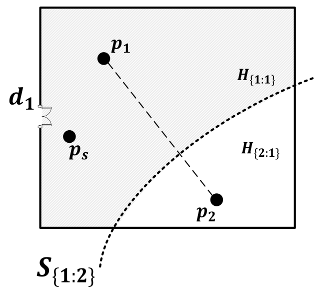


Figure 2: Example of region divided by a splitter

Definition 4.1 (Splitter). Given a hyperbola of $|d(P, F_i) - d(P, F_j)| = 2a$. We identify one of the curves as a splitter that divides the space into two half spaces $d(P, F_i) - d(P, F_j) < 2a$ and $d(P, F_i) - d(P, F_j) > 2a$, denoted by $\mathcal{S}_{i:j}$ where i and j are the two foci. The corresponding vertex that the curve goes through is called the split point.

Figure 2 depicts an indoor partition I with door d_1. The partition consists of two indoor points p_1, p_2. Assume that the partition I is the only candidate partition in the indoor space. Hence, the minimum detour must be one of the local detours passes through the door d_1 , i.e., $D^L_{1:1} = \{p_s \rightarrow p_1 \rightarrow d_1 \rightarrow p_t\}$ and $D^L_{2:1} = \{p_s \rightarrow p_2 \rightarrow d_1 \rightarrow p_t\}$.

Let $H_{1:1}$ and $H_{2:1}$ be the regions inside the indoor partition where $D^L_{1:1}$ and $D^L_{2:1}$ are minimum detours respectively. When p_s lies in the region $H_{1:1}$, the following inequality must be satisfied.

$$L(D^L_{1:1}) < L(D^L_{2:1})$$
$$d(p_s, p_1) + d(p_1, d_1) + d(d_1, p_t) < d(p_s, p_2) + d(p_2, d_1) + d(d_1, p_t)$$
$$d(p_s, p_1) - d(p_s, p_2) < d(p_2, d_1) - d(p_1, d_1)$$

According to the Definition 4.1, we can construct a splitter (i.e., $\mathcal{S}_{1:2}$) by selecting the indoor points p_1, p_2 as foci and $2a = d(p_2, d_1) - d(p_1, d_1)$. The most important step in constructing a splitter is to determine the split point correctly. Let point x be the split point and point c be the center. Then the split point is determined as follows,

$$\begin{aligned} d(p_1, x) - d(p_1, c) &= a \\ d(p_1, x) - \frac{d(p_1, p_2)}{2} &= \frac{1}{2}\Big(d(p_2, d_1) - d(p_1, d_1)\Big) \\ d(p_1, x) &= \frac{1}{2}\Big(d(p_1, p_2) + d(p_2, d_1) - d(p_1, d_1)\Big) \end{aligned} \tag{2}$$

Once the splitter $\mathcal{S}_{1:2}$ is constructed, space is divided into two half spaces $d(p, p_1) - d(p, p_2) < 2a$ (i.e., the shaded area in Figure 2) and $d(p, p_1) - d(p, p_2) > 2a$ which is basically the required $H_{1:1}$ and $H_{2:1}$ regions respectively. Intuitively, these two regions act as safe zones where we can guarantee that the minimum local detour does not change until the current location of the user is within a particular region. Moreover, as the Equation (2) depicts, the splitter $\mathcal{S}_{1:2}$ is independent of the query parameters. Hence, these two regions can be pre-computed and utilized in query time to quickly identify the minimum local detour that goes through the door d_1 with respect to a location update inside the partition I.

Furthermore, for the candidate partitions with indoor points more than two, we need to construct splitters for each possible pair of indoor points to determine such safe zones. Note that these regions are AW Voronoi cells with additive weight $d(d_i, p_j)$ which is the distance between indoor point p_j and door d_i. Hence, in the pre-processing approach, we generate AW Voronoi diagram for each partition door d_i. In query processing, we can easily determine the minimum local detour that goes through a particular door by looking at the corresponding AW Voronoi cell with respect to the current user location. For the indoor partitions that have more than one door, AW Voronoi diagram for each door d_i is created with the additive weight $d(d_i, p_i)$. For such a partition, the local computation is done as follows. First, the minimum local detour points with respect to each door is obtained using corresponding AW Voronoi diagrams. Then, these results are evaluated to determine the minimum local detour point. We observed that the number of doors of a candidate partition is very small in real-world applications, i.e., at most 3-4 doors. Thus, the local computation can be done efficiently.

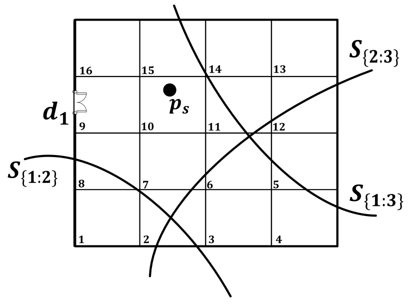


Figure 3: Example of 4×4 grid

4.1.2 Safe zone Indexing. As we have already stated, AW Voronoi diagrams for each candidate partition is determined by constructing the splitters between the indoor points. Since these Voronoi cells are formed by curved splitters, such an AW Voronoi diagram cannot be easily indexed unless they are approximated using polygons. But it incurs inaccurate results in query time. Hence, we accompanied an approach that index AW Voronoi diagrams by utilizing the grid data structure. In our grid index, the space of the corresponding indoor partition is subdivided into $2^n \times 2^n$ grid cells (where $n > 0$) as shown in Figure 3. Then in each grid cell, we store the splitters that overlap with the cell. To record these overlapping splitters in each grid cell, we used conceptual tree-based grid access method [6] where we consider the root of the conceptual tree is a rectangle covering the whole indoor partition. Then the root cell is divided into four equal grid cells that represent the next level of the tree. The process continues until each entry of the leaf level represents one grid cell. During the process, for an intermediate cell, we take into account the splitters overlaps with the cell and only these splitters are considered when marking the child cells of the particular cell. By doing this, we significantly improve the process of constructing the AW Voronoi diagrams for a given partition.

Figure 3 shows an example of AW Voronoi diagram (which is a 4×4 grid) for door d_1 of the partition I . The grid cell with cell ID 2 will be marked with $S_{1:2}$ and $S_{2:3}$ as they overlap with the particular grid cell. Similarly, grid cell with cell ID 11 is marked with $S_{1:3}$ and $S_{2:3}$. The grid cells with cell IDs 1, 3, 9, 13, 16 remains empty as they are not overlap by a splitter which indicates that these grid cells are completely inside AW Voronoi cells. Thus, the

Algorithm 1: *getRemoteDetour*(d_k, p_t)

```
   Data: VIP-tree $\mathcal{V}$, a door $d_k$, an indoor point $p_t$
   Result: minimum remote detour $D^R$
1  $p \leftarrow \emptyset$; $D \leftarrow \emptyset$
2  $Q.enqueue(\mathcal{V}.root, 0)$;
3  while $Q$ is not empty do
4      $element \leftarrow Q.dequeue()$;
5      if $element$ is a point then
6          $p \leftarrow element$;
7      end
8      else if $element$ is a partition then
9          foreach point $p_a$ in $element$ do
10             foreach $\{d_i, d_j\}$ door combination do
11                 $minCost \leftarrow$
                     $d(d_k, d_i) + d(d_j, p_t) + d(d_i, p_k) + d(p_k, d_j)$;
12                 $Q.enqueue(p_a, minCost)$;
13             end
14         end
15     end
16     else
           // element is a tree node
17         foreach child-node $N$ of $element$ do
18             $minCost \leftarrow d(d_k, N) + d(N, p_t)$;
19             $Q.enqueue(N, minCost)$;
20         end
21     end
22 end
23 $D^R \leftarrow \{d_i, p, p_t\}$
24 return $D^R$;
```

minimum local detour point (i.e., the corresponding indoor point of the Voronoi Cell) for a user location within these grid cells can be quickly determined. For the grid cells with overlapping splitters, the minimum local detour point is determined by evaluating corresponding overlapping splitters.

4.2 Remote Computation

In this section, we present an efficient approach to compute results for our second subproblem which is determining the minimum remote detour with respect to the user's location (denoted as "remote computation"). Intuitively, for any user location inside an indoor partition, the minimum remote detour must be a remote detour that starts from one of the partition doors. Hence, we propose a best first search algorithm *getRemoteDetour()*, i.e., Algorithm 1, that retrieves the minimum remote detour from a given door to the query target point, i.e., p_t. This algorithm accesses the VIP-tree components, i.e., tree nodes, partitions and indoor points, based on the smallest aggregated indoor distance from the given door and the query target point to retrieve the minimum remote detour point. After obtaining the minimum remote detour points for each partition door, the minimum remote detour with respect to the user location can be readily determined. Moreover, if the next location update is within the same partition, then we can reuse these remote detour results to determine the minimum remote detour for the new user location. As Algorithm 1 illustrates, we traverse each level of the VIP-tree starting from the root node (line 2) and compute the detour costs for tree nodes at each level. We terminate the algorithm when the min-priority queue Q is empty (line 3) or the minimum detour point is found (line 5-6). We enqueue each element with the detour distance as its key value. In each iteration we dequeue the element with the minimum key value, i.e., smallest detour distance. If the dequeued element is a node we enqueue all the child nodes (line 16-21). If the element is an indoor partition then we enqueue all the indoor points inside the partition. Note that, an indoor point may be enqueued multiple times into the queue as the detours through different door combination are possible (line 8-15). When the dequeued element is an indoor point, the algorithm is terminated and the indoor point along with its detour distance is returned (line 24).

4.3 Query Processing

Now we proceed to explain our solution that utilizes the results of the two subproblems, i.e., local and remote computations, to determine the minimum detour for a given detour query. When the sytem is initiated, the AW Voronoi diagrams of all the partitions are loaded into the server memory. Thus, the AW Voronoi diagram for an indoor partition can be accessed efficiently in the query processing. As Algorithm 2 illustrates, first, we initialize I with the current partition (line 3). Next we compute the remote detours for each door of the partition I (line 5). Meanwhile, we update the minimum remote detour with respect to the current user location by selecting the remote detour with the smallest detour distance (line 6-7). Then, we check whether the current partition, i.e., I, is a candidate partition (line 10). If so we must perform the local computation to obtain the minimum local detour point. Thus, we utilize each of the AW Voronoi diagrams of partition doors to compute the minimum local detour. Initially, the grid cell that the current user location p_s lies is identified. Then, the splitters overlaps with the corresponding grid cell are retrieved (line 14). Also, we store these splitters using a list, i.e., *sList*, (in Section 4.3.1, we explain the purpose of storing these splitters). Next, the minimum local detour point is obtained by evaluating the overlapping splitters. (line 12-14). After determining the local detour points per door, we generate the corresponding local detours to find the one with minimum detour distance among them (line 15-17). Note that, the shortest path distances from door d_i to target point p_t, i.e., $dt(d_i, p_t)$, must be determined to compute the local detour distances. Then the safe zone for the user is constructed (line 21). Note that the safe zone for the user is computed using the indexed safe zones of the indoor points and the detail of the construction of such a safe zone for the user is discussed in Section 4.3.1. The minimum detour is determined by selecting the minimum of the remote and local detours (line 22). Finally, the minimum detour and the safe zone are returned (line 23).

4.3.1 Safe zone Construction for Users. In order to reduce the communication cost, we send a safe zone such that the client device can monitor the query results by itself without contacting the server. Even though the actual safe region for a minimum local detour via a door is the corresponding AW Voronoi cell, we assign the area of the grid cell that user located in. The reason behind this is to reduce the workload at the client-side by sending only a small number of boundaries to monitor (Note that an AW Voronoi cell may consist of a large number of boundaries.) Since a grid cell may belong to different AW Voronoi cells, the overlapping splitters of the particular cell are sent to the user to determine the minimum

Algorithm 2: Query Processing

```
Data: VIP-tree $\mathcal{V}$, a query $q = \{p_s, p_t\}$
Result: Minimum Detour $D$, Safe zone $Z$
1  $D^R \leftarrow \emptyset$; $D^L \leftarrow \emptyset$; $sList \leftarrow \emptyset$;
2  $I \leftarrow$ current partition;
   // Remote Computation
3  foreach $d_i$ door of $I$ do
4      $D_i^R \leftarrow getRemoteDetour(d_i, p_t)$ ;            // Algorithm 1
5      if $d(p_s, d_i)+L(D_i^R) < L(D^R)$ then
6          $D^R \leftarrow D_i^R$ ;      // minimum remote detour w.r.t $p_s$
7      end
8  end
   // Local Computation
9  if $I$ is a candidate partition then
10     foreach $d_i$ door of $I$ do
11         $Cell \leftarrow$ the corresponding grid cell
12         $sList \leftarrow sList \cup$ the set of splitters overlaps with $Cell$;
13         $p_j \leftarrow$ local detour point for $d_i$
14         $L(D_i^L) \leftarrow \{p_s \rightarrow p_j \rightarrow d_i \rightarrow d_t\}$;
15         if $L(D_i^L) < L(D^L)$ then
16             $D^L \leftarrow D_i^L$ ;   // minimum local detour w.r.t $p_s$
17         end
18     end
19 end
20 construct $Z$ ;                                        // Section 4.3.1
21 $D \leftarrow min\{D^R, D^L\}$;
22 return $D$, $Z$;
```

local detour point for the next location update without communicating the server. Moreover, the distances of the shortest paths from each partition door are also sent to assist the client-side local computation. In addition to that, the remote detours for each door are sent to the user for the client-side remote computation. Thus, the client device can compute the query result for the next location update without communicating with the server if the user is still inside the safe zone. Note that, for a non-candidate partition, only the remote computation is required. Thus, we assign the whole space of the indoor partition as the safe zone and send the materialized remote detours to support the client-side remote computation. Hence, the client device can determine the minimum detour for any location update within the particular partition without contacting the server.

4.4 Continuous Monitoring

The result of the query needs to be continuously monitored since the user is continuously moving and user movements may change the query result. Naively, the server can recompute the query result at each time the server receives a location update from the user. Since the indoor venues consist of thousands of indoor points, these re-computations are very expensive. In the experiments, we show the performance difference between this naive approach and our solution. As we already stated, some initial computations for an indoor partition can be reused until the user leaves the particular partition. Hence, we materialize the minimum remote detours (line 5 in Algorithm 2) and shortest path distances (line 16 in Algorithm 2) for each door of the current indoor partition. Then, the query result

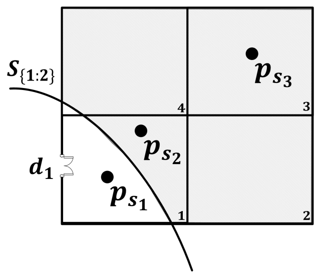


Figure 4: Example of continuous monitoring

computation for any location update about to happen inside the current partition can be handled efficiently.

In continuous monitoring, first, the server checks whether the user is still in the same partition so that the server can reuse the materialized data to determine the minimum detour. The server uses the materialized remote detours to compute the minimum remote detour for the location update. Then the server accesses AW Voronoi diagrams to figure out the minimum local detour points for each door of the partition and determines the minimum local detour. Finally, the server sends the safe zone along with the minimum detour to reduce the communication cost occur due to frequent location updates. With the help of the safe zone constructed for the user, the client device can monitor the query result without communicating the server.

In the client side, we assume that the client device is capable of storing all the information that is sent by the server, i.e., the remote detours, the shortest path distances and the safe zone, and computing minimum detour for any user movement within the safe zone. Once the user moves outside the safe zone, it sends a location update to the server. Then the server computes the query result for the new location and send the result along with the safe zone. For example, Figure 4 shows a 4×4 grid of a candidate partition with door d_1. The area shaded in grey is the AW Voronoi cell of the indoor point p_1 and white area is the AW Voronoi cell of the indoor point p_2. Let the user is at p_{s_1} position. Thus, the area of the grid cell 1 is allocated as the safe zone for the user. Assume that the user moves to p_{s_2}. Since the user is still within the safe zone, the client device will determine the local minimum detour point as p_1 by evaluating the splitter $S_{1:2}$. Moreover, the client device will perform the remote computation by utilizing materialized data to determine the minimum detour without communicating with the server. For the next movement, i.e., p_{s_3}, the client device will send a location update to the server since the user has leaf the safe zone.

4.5 Local Computation without AWVDs

As indoor venues consist of a considerable number of indoor points, most of the indoor partitions are highly populated with indoor points. Hence, a large amount of possible local detours must be generated to determine the minimum local detour. We develop a competitive approach called "local" which is an improvement of the naive approach. Similar to our approach, the local approach utilizes the materialized data such as the remote detours and the distances of the shortest paths from each partition door such that they are utilized in the next location updates happen within the current partition. Besides, the local approach generates all possible local detours to determine the minimum local detour and send only the result to the user. Thus, the communication overhead remains the same as the naive approach. The empirical study shows that our solution performs much better than this approach.

5 EXPERIMENTS

5.1 Experimental Settings

5.1.1 Indoor Space Datasets. We used two indoor space datasets for our experiments. One of them is the real-world dataset [14] of the largest shopping centre in Australia. The dataset consists of over 300 indoor partitions that are spread over 4 levels. This dataset was manually converted into machine-readable indoor venues and the sizes of indoor partitions (e.g., rooms, hallways) were determined using OpenStreetMap [2]. We denote this dataset by CHAD. The D2D graph for this indoor space consists of 338 vertices (i.e., doors) and 3847 edges. We synthetically generated five indoor object datasets that have 1K, 2K, 3K, 4K and 5K indoor points, respectively. Each indoor point was randomly selected inside a partition of the indoor space.

The other dataset is a replica of the CHAD dataset (denoted as CHAD-2). It was obtained by placing a replica of Chadstone Shopping Centre on top of the original building. This dataset consists of 678 rooms and the D2D graph for this indoor space consists of 676 vertices (i.e., doors) and 7698 edges. We again generated five indoor object datasets for this indoor space by randomly selecting 5K, 10K, 15K, 20K and 25K indoors points, respectively. Moreover, we generated 100 queries for each experimental setting. The source and target indoor points of each query were randomly determined.

5.1.2 Trajectory Datasets. We created synthetic trajectory datasets for all our experiments. To generate a trajectory, we start from the query source point and randomly pick an indoor point inside a candidate partition. By doing this, we make sure that the user trajectories pass through candidate partitions. Also, these random points are selected in a way that the trajectory leads towards the query target point to demonstrate the real-world scenarios. After determining a random indoor point, we let the user moves towards that point with a particular speed. We continued this for 500 timestamps, by determining a new random point similarly as the user reaches one. The walking speeds of the users are determined as follows. We chose 0.5, 1.5 and 2.5 meters per timestamp as the user speeds in generating the different trajectory datasets. Furthermore, we denote these speeds by slow, medium and fast respectively in the later discussions.

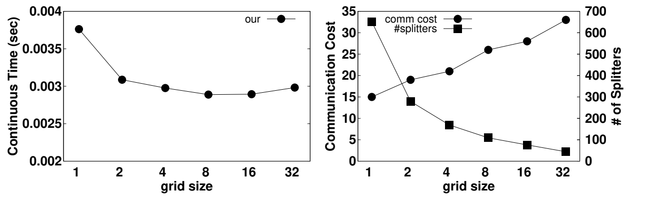
5.1.3 Competitors. We compare our algorithm with two competitors. First one is the naive approach which computes the query result from scratch for each location update. We denote this by "naive". Our second competitor is the approach mentioned in Section 4.5 denoted by "local" which is an improvement of the naive approach that does only the local computation naively for each location update.

All algorithms were implemented in C++ and our experiments were conducted on Ubuntu running on an Intel Core i5 @ 3.30GHz and 4GB RAM.

[2]https://www.openstreetmap.org/

Table 2: The parameters used for experiments

Parameter	Default	Range
# indoor points	3K	1K, 2K, 3K, 4K, 5K
speed	medium	slow, medium, fast
grid size	8	1, 2, 4, 8, 16, 32



(a) (b)

Figure 5: Varying grid size on the CHAD dataset

5.2 Experimental Results

In all the experiments, we use the default settings (see Table 2) while varying a single parameter at a time. First, we conduct the set of experiments on CHAD dataset and then on CHAD-2 dataset which is the replica of the real-world dataset. Moreover, for each experiment, we report the average continuous time in milliseconds.

5.2.1 Varying the grid size. As mentioned in Section 4.1.2, we use the grid indexing structure to index the AW Voronoi diagrams for efficient local computation in query processing. As Figure 5(a) illustrates, our solution performs well (which is less than 0.003 seconds) when the grid size is 8. Figure 5(b) reports the communication cost and the number of splitters which are basically the average number of times that the client device communicates with the server and the average number of splitters monitored by the system respectively. As Figure 5(b) illustrates, the communication cost can be significantly reduced by selecting the grid size as 1. The reason is that the whole space of indoor partition is assigned as the safe zone. Even though small grid sizes give low communication cost as large area is assigned as safe zones, the client side computational cost increases since the number of splitters monitored by the client device increases accordingly. For example, even though the average communication cost for grid size 1 is 15, the client device has to monitor more than 650 splitters in the local computation. We selected the grid size 8 as the default gird size as it gives the best performance for our solution. For grid size 8, the communication overhead is 25 which is 20 times better than the competitors. And also, the system has to monitor approximately 100 splitters per detour query.

Moreover, we report the indexing cost of AW Voronoi diagrams in megabytes. As Figure 6 illustrates, the amount of memory required to store the AW Voronoi diagrams increases as the grid size increases because the number of cells increases exponentially. But the required memory spaces for small grid sizes are feasible. For the default grid size which is 8, the AW Voronoi diagrams of CHAD and CHAD-2 datasets need only 117MB and 567MB memory space respectively.

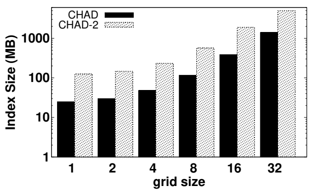


Figure 6: Indexing Cost

5.2.2 Varying the number of indoor points. This set of experiments is done to evaluate all the algorithms on the CHAD dataset. First, we investigate the continuous time of the algorithms by varying the number of points (i.e., indoor objects) in the indoor venue. As Figure 7(a) shows, the continuous times of all algorithms increase as the number of indoor points are increased. The reason is both remote and local detour computation costs increase as the number possible detours increases. Even though our solution does not consider all possible detours in local computation, still the continuous time increases accordingly as the number of splitters to be monitored increases. For the default settings, our algorithm is 15 times better than the local approach while three orders of magnitude better than the naive approach. As the competitors do not use the safe zone concept, in terms of the communication overhead we are always better since they have to communicate the server to get the results. The continuous time of naive drastically increases as it does both remote and local computation for each location update. Clearly, our solution outperforms the competitors under all the settings.

Figure 7(b) reports the average number of indoor points accessed by the local approach and the average number of splitters monitored by our approach. It is obvious that both values increases as the indoor points in the indoor venue increases. Moreover, the reported results clearly explain the performance degradation of both our and local approaches in the experiments shown in Figure 7(a).

5.2.3 Varying the speed. Next, we evaluate the algorithms by varying the walking speed of the user. As Figure 8(a) shows, the continuous times of all algorithms increase when the speed is increased. It is obvious for our and local solutions that the continuous time increases as the user quickly leaves partition due to the higher walking speed. Note that, for our solution, the user stays inside a safe zone only for a small amount of time. The reasons for this performance degradation of the naive solution is that the user is visiting more candidate partition when the speed is increased. Thus, local and remote detour computations increases.

Figure 8(b) reports the average number of rooms visited by the user and the average number of indoor points accessed by the local algorithm by varying the speed of the user. When the user's speed is fast, the user visits 30 rooms while only 5 rooms when speed is slow. As we mentioned earlier, clearly, the continuous times of all algorithms increase as the speed increased since the user is quickly leaving partitions. Also, the number of candidate partitions visited by the user is increased as more rooms are visited. Hence, the continuous times of both our and local approaches increases as the number of indoor points accessed and the number of splitters monitored increase.

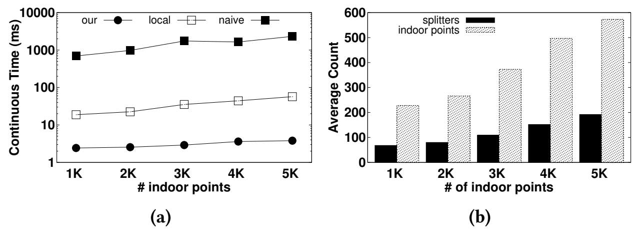


Figure 7: Varying the number of indoor points

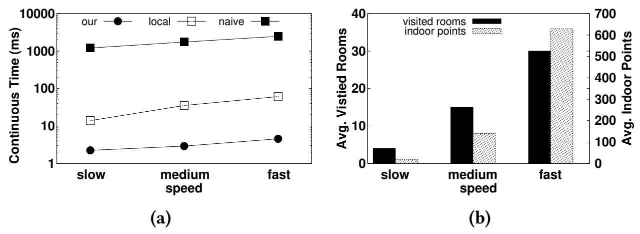


Figure 8: Varying the speed of the user

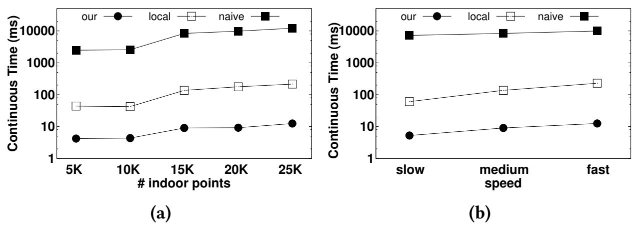


Figure 9: Results on the CHAD-2 dataset

5.2.4 Experiments on a large dataset. The objective of this set of experiments is to study the scalability of the proposed solution. Hence, we use the CHAD-2 dataset. Since this dataset has a large number of indoor points, the number of candidate partitions for this dataset is higher than the real-world dataset. Figure 9(a) shows the continuous times of algorithms varying the number of indoor points. The performances of all the algorithms decrease when the number of indoor points is increased. Clearly, our approach is three order of magnitude better than naive solution when the dataset consists of 25K indoor points. And also 20 times better than the local approach. Moreover, Figure 9(b) reports continuous times by varying the user speed on CHAD-2 dataset. Under all the settings, our approach performs better than competitive approaches. The performance of all algorithms has decreased compared to the results on CHAD dataset due to a large number of indoor points and candidate partitions. Note that, our approach is still able to answer a query in a reasonable time. These results conclude that our approach offers scalability and great performance.

5.2.5 Effectiveness of safe zones. Figure 10 reports the escape probability of our solution varying the users' speed. The escape probability is determined by dividing the number of times user communicate with the server by the total number of movements. As expected,

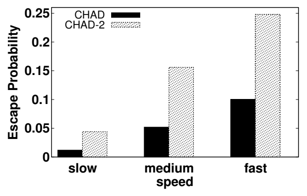


Figure 10: Escape Probability

the escape probability increases with the user speed. The reason is, the users leave safe zones very quickly due to the speed. The escape probability is 0.1 in CHAD dataset while 0.25 in CHAD-2 dataset when the user is moving fast. The reason behind the high escape probability of the CHAD-2 dataset is that the number of candidate partitions in the CHAD-2 dataset is large. Hence, the users pass through many candidate partitions in their trajectories. The results conclude that the proposed solution is effective in real-world applications since the escape probability is small.

6 CONCLUSION

In this paper, we propose an efficient solution to answer continuous detour queries in the indoor space. First, we introduce a pre-processing approach for efficient local computation that constructs safe zones for indoor objects. Then, we propose a best first search algorithm to efficiently compute a remote detour for a given door of an indoor partition. Finally, we integrate the outcome of the local and remote computations and introduce a client-server framework to answer the continuous detour queries efficiently. The results of the empirical studies show that our approach outperforms the naive approach by at least three orders of magnitude while average 15 times faster than the improved naive approach. Also, our approach reduces the communication cost.

ACKNOWLEDGMENTS

Muhammad Aamir Cheema is supported by Australian Research Council (ARC) FT180100140 and DP180103411.

REFERENCES

[1] Tenindra Abeywickrama, Muhammad Aamir Cheema, and Arijit Khan. 2019. K-SPIN: Efficiently Processing Spatial Keyword Queries on Road Networks. *IEEE Transactions on Knowledge and Data Engineering* (2019).

[2] Tenindra Abeywickrama, Muhammad Aamir Cheema, and David Taniar. 2016. k-Nearest Neighbors on Road Networks: A Journey in Experimentation and In-Memory Implementation. *PVLDB* 9, 6 (2016), 492–503. https://doi.org/10.14778/2904121.2904125

[3] Muhammad Aamir Cheema. 2018. Indoor location-based services: challenges and opportunities. *SIGSPATIAL Special* 10, 2 (2018), 10–17. https://doi.org/10.1145/3292390.3292394

[4] Muhammad Aamir Cheema, Ljiljana Brankovic, Xuemin Lin, Wenjie Zhang, and Wei Wang. 2010. Multi-guarded safe zone: An effective technique to monitor moving circular range queries. In *Data Engineering (ICDE), 2010 IEEE 26th International Conference on.* IEEE, 189–200.

[5] Muhammad Aamir Cheema, Ljiljana Brankovic, Xuemin Lin, Wenjie Zhang, and Wei Wang. 2011. Continuous Monitoring of Distance-Based Range Queries. *IEEE Trans. Knowl. Data Eng.* 23, 8 (2011), 1182–1199. https://doi.org/10.1109/TKDE.2010.246

[6] Muhammad Aamir Cheema, Xuemin Lin, Ying Zhang, Wei Wang, and Wenjie Zhang. 2009. Lazy updates: An efficient technique to continuously monitoring reverse knn. *Proceedings of the VLDB Endowment* 2, 1 (2009), 1138–1149.

[7] Zaiben Chen, Heng Tao Shen, Xiaofang Zhou, and Jeffrey Xu Yu. 2009. Monitoring path nearest neighbor in road networks. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data.* ACM, 591–602.

[8] Ken CK Lee, Wang-Chien Lee, Baihua Zheng, and Yuan Tian. 2012. ROAD: A new spatial object search framework for road networks. *IEEE transactions on knowledge and data engineering* 24, 3 (2012), 547–560.

[9] Hua Lu, Xin Cao, and Christian S Jensen. 2012. A foundation for efficient indoor distance-aware query processing. In *Data Engineering (ICDE), 2012 IEEE 28th International Conference on.* IEEE, 438–449.

[10] Hua Lu, Bin Yang, and Christian S Jensen. 2011. Spatio-temporal joins on symbolic indoor tracking data. In *Data Engineering (ICDE), 2011 IEEE 27th International Conference on.* IEEE, 816–827.

[11] Sarana Nutanong, Egemen Tanin, Jie Shao, Rui Zhang, and Ramamohanarao Kotagiri. 2012. Continuous detour queries in spatial networks. *IEEE Transactions on Knowledge and Data Engineering* 24, 7 (2012), 1201–1215.

[12] Atsuyuki Okabe, Barry Boots, Kokichi Sugihara, and Sung Nok Chiu. 2009. *Spatial tessellations: concepts and applications of Voronoi diagrams.* Vol. 501. John Wiley & Sons.

[13] Rudra Ranajee Saha, Tanzima Hashem, Tasmia Shahriar, and Lars Kulik. 2018. Continuous Obstructed Detour Queries. In *10th International Conference on Geographic Information Science (GIScience 2018).* Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

[14] Chaluka Salgado, Muhammad Aamir Cheema, and David Taniar. 2018. An efficient approximation algorithm for multi-criteria indoor route planning queries. In *Proceedings of the 26th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems.* ACM, 448–451.

[15] Shuo Shang, Ke Deng, and Kexin Xie. 2010. Best point detour query in road networks. In *Proceedings of the 18th SIGSPATIAL International Conference on Advances in Geographic Information Systems.* ACM, 71–80.

[16] Zhou Shao, Muhammad Aamir Cheema, and David Taniar. 2017. Trip Planning Queries in Indoor Venues. *Comput. J.* 61 (2017), 1–18.

[17] Zhou Shao, Muhammad Aamir Cheema, David Taniar, and Hua Lu. 2016. VIP-Tree: an effective index for indoor spatial queries. *Proceedings of the VLDB Endowment* 10, 4 (2016), 325–336.

[18] Shashi Shekhar and Jin Soung Yoo. 2003. Processing in-route nearest neighbor queries: a comparison of alternative approaches. In *Proceedings of the 11th ACM international symposium on Advances in geographic information systems.* ACM, 9–16.

[19] Wei-Wei Sun, Chu-Nan Chen, Liang Zhu, Yun-Jun Gao, Yi-Nan Jing, and Qing Li. 2015. On efficient aggregate nearest neighbor query processing in road networks. *Journal of computer science and technology* 30, 4 (2015), 781–798.

[20] Xike Xie, Hua Lu, and Torben Bach Pedersen. 2013. Efficient distance-aware query evaluation on indoor moving objects. In *Data Engineering (ICDE), 2013 IEEE 29th International Conference on.* IEEE, 434–445.

[21] Xike Xie, Hua Lu, and Torben Bach Pedersen. 2015. Distance-aware join for indoor moving objects. *IEEE Transactions on Knowledge and Data Engineering* 27, 2 (2015), 428–442.

[22] Bin Yang, Hua Lu, and Christian S Jensen. 2009. Scalable continuous range monitoring of moving objects in symbolic indoor space. In *Proceedings of the 18th ACM conference on Information and knowledge management.* ACM, 671–680.

[23] Bin Yang, Hua Lu, and Christian S Jensen. 2010. Probabilistic threshold k nearest neighbor queries over moving objects in symbolic indoor space. In *Proceedings of the 13th international conference on extending database technology.* ACM, 335–346.

[24] Shiyu Yang, Muhammad Aamir Cheema, Xuemin Lin, Ying Zhang, and Wenjie Zhang. 2017. Reverse k nearest neighbors queries and spatial reverse top-k queries. *VLDB J.* 26, 2 (2017), 151–176. https://doi.org/10.1007/s00778-016-0445-2

[25] Man Lung Yiu, Nikos Mamoulis, and Dimitris Papadias. 2005. Aggregate nearest neighbor queries in road networks. *IEEE Transactions on Knowledge and Data Engineering* 17, 6 (2005), 820–833.

[26] Jin Soung Yoo and Shashi Shekhar. 2005. In-route nearest neighbor queries. *GeoInformatica* 9, 2 (2005), 117–137.

[27] Wenjie Yuan and Markus Schneider. 2010. Supporting continuous range queries in indoor space. In *Mobile Data Management (MDM), 2010 Eleventh International Conference on.* IEEE, 209–214.

[28] Pengfei Zhang, Huaizhong Lin, Yunjun Gao, and Dongming Lu. 2018. Aggregate keyword nearest neighbor queries on road networks. *GeoInformatica* 22, 2 (2018), 237–268.

[29] Ruicheng Zhong, Guoliang Li, Kian-Lee Tan, and Lizhu Zhou. 2013. G-tree: An efficient index for knn search on road networks. In *Proceedings of the 22nd ACM international conference on Information & Knowledge Management.* ACM, 39–48.

[30] Ruicheng Zhong, Guoliang Li, Kian-Lee Tan, Lizhu Zhou, and Zhiguo Gong. 2015. G-tree: An efficient and scalable index for spatial search on road networks. *IEEE Transactions on Knowledge and Data Engineering* 27, 8 (2015), 2175–2189.

[31] Liang Zhu, Yinan Jing, Weiwei Sun, Dingding Mao, and Peng Liu. 2010. Voronoi-based aggregate nearest neighbor query processing in road networks. In *Proceedings of the 18th SIGSPATIAL International Conference on Advances in Geographic Information Systems.* ACM, 518–521.