An Efficient Approach for Indoor Facility Location Selection

Yeasir Rayhan
Bangladesh University of Engineering and Technology
Dhaka, Bangladesh
yeasirrayhanprince@gmail.com

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

Muhammad Aamir Cheema
Faculty of Information Technology, Monash University
Australia
aamir.cheema@monash.edu

Hua Lu
Department of People and Technology, Roskilde University
Roskilde, Denmark
luhua@ruc.dk

Mohammed Eunus Ali
Bangladesh University of Engineering and Technology
Dhaka, Bangladesh
eunus@cse.buet.ac.bd

ABSTRACT

The advancement of indoor location-aware technologies enables a wide range of location based services in indoor spaces. In this paper, we formulate a novel *Indoor Facility Location Selection (IFLS)* query that finds the optimal location for placing a new facility (e.g., a coffee station) in an indoor venue (e.g., a university building) such that the maximum distance of all clients (e.g., staffs/students) to their nearest facility is minimized. To the best of our knowledge we are the first to address this problem in an indoor setting. We first adapt the state-of-the-art solution in road networks for indoor settings, which exposes the limitations of existing approaches to solve our problem in an indoor space. Therefore, we propose an efficient approach which prunes the search space in terms of the number of clients considered, and the total number of facilities retrieved from the database, thus reducing the total number of indoor distance calculations required. The key idea of our approach is to use a single pass on a state-of-the-art index for an indoor space, and reuse the nearest neighbor computation of clients to prune irrelevant facilities and clients. We evaluate the performance of both approaches on four indoor datasets. Our approach achieves a speedup from 2.84× to 71.29× for synthetic data and 97.74× for real data over the baseline.

1 INTRODUCTION

The problem of finding the optimal location has been widely studied for both Euclidean space and road networks [2–4, 9, 12, 14, 21, 22, 24]. Given a set of clients and a set of existing facilities, the problem requires identifying the optimal candidate location for the new facility that optimizes an objective function. Existing works have mainly focused on optimizing three objective functions. MinDist function minimizes the average distance of the clients from their nearest facilities, MinMax function minimizes the maximum distance of the clients from their nearest facilities, and MaxSum maximizes the number of the clients who will have the new facility as the nearest one. Most of the works [2, 5, 12, 24, 26] identify the optimal location from the continuous space and few works [3, 4, 8] select it from a set of given candidate locations. Nevertheless, to the best of our knowledge the problem of finding the optimal location has not been addressed in indoor settings. Furthermore, as shown in [19], the techniques designed for outdoor space cannot be effectively extended for indoor space due to its unique topology and characteristics. On top of that, a brute force approach is also not desirable specially in the context of dynamic crowd scenarios (e.g., changing crowd), where the position a new facility needs to be updated constantly or an additional new facility needs to be established instantly.

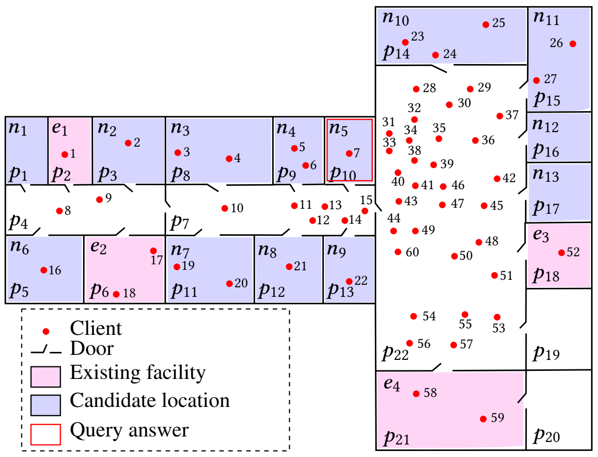


Figure 1: Indoor Facility Selection Query.

In this paper, we focus on finding the optimal location for a new facility from a set of given candidate locations for a set of clients in an indoor venue so that the maximum indoor distance of the clients from their nearest facilities is minimized. There are a number of real-life applications for the proposed indoor facility location selection query. For example, a hospital may want to identify a location to set up a new nurse station from a set of candidate locations such that it minimizes the maximum indoor distance between the patient beds and their nearest nurse stations. Similarly, a university authority may want to find a location to place a new facility (e.g., printer, coffee or vending machine) that minimizes the maximum indoor distance between the students/staffs and their nearest facility. Again, an advertising agency may want to place their advertising booth in a shopping mall and there may be restrictions on where such booths can or cannot be installed. Thus, given a set of candidate locations, the goal is to find the booth location that maximises the coverage.

Figure 1 shows an example of an indoor space with 22 partitions (p_1 to p_{22}) and 60 clients (c_1 to c_{60}). For clarity, we avoid using c to represent the clients in the figure. Let us assume that we want to open a new *coffee* facility in one of the partitions from n_1 to n_{13} in this indoor space. There already exist four *coffee* facilities (e_1 to e_4). The indoor facility location selection query returns the location $n_5(p_{10})$ for establishing the new coffee facility since it optimizes the MinMax objective function, i.e., establishing a new *coffee* facility at n_5 will minimize the maximum indoor distance of all the clients from their nearest coffee facilities.

Existing work for outdoor space [2] considers MinMax function and selects the optimal location in road networks. We cannot directly apply the state-of-the-art work [2] for our purpose as the construction of indoor space is different from road networks. More specifically, in road networks, a user's movement is restricted on roads, whereas in the indoor venues, the users can freely move inside a partition but the movement among the partitions are restricted through doors and stairs. We adapt the most efficient state-of-the-art technique [2] to find new facility location in an indoor space as a baseline solution, which helps identifying several potential limitations of the current state-of-the-art.

Major computational challenges to evaluate an indoor facility selection query are finding the nearest existing facility and nearest candidate locations for a large number of clients, and the huge number of indoor distance computations. We propose an efficient approach that addresses the above issues by (i) refining the facility search space, (ii) pruning the clients, and (iii) finding nearest neighbours for clients incrementally in the indoor space with a reduced number of indoor distance computations.

Our approach does not need to retrieve any facility (existing or candidate) outside the refined search space. This allows us to avoid the computation of the clients' nearest existing facilities not in the refined search space. Moreover, we identify and prune the clients for whom the new facility cannot improve their distance from the existing facilities. Our proposed efficient algorithm incrementally finds the nearest neighbours from both existing facilities and candidate locations for new facilities in the indoor space for all clients with a single search. Our algorithm significantly reduces the number of indoor distance computations by grouping the clients while exploring the facilities. On average, our efficient approach achieves a speedup from 2.84× to 71.29× for synthetic data and 97.74× for real data over the baseline .

Our contributions in the paper are summarized as follows.

- To the best of our knowledge, we are the first to formulate the facility location selection query in an indoor space.
- We devise a non-trivial baseline solution by modifying the state-of-the-art solution designed for road networks, and highlight the baseline's limitations for indoor spaces.
- We propose an efficient approach for solving the indoor facility selection queries that prune the data space both w.r.t. the facilities and clients in a single pass in the database.
- We run an extensive set of experiments on four real indoor venues, namely Melbourne Center, Chadstone shopping center, Copenhagen Airport, and Menzies Building using both synthetic and real dataset.

2 RELATED WORK

2.1 Non-Indoor Facility Location Selection

Even though the facility location selection queries have been studied in a wide range of applications in literature, the underlying idea of the problem remains same: given a set of objects (clients) and facilities, find a facility from a discrete or continuous space which optimizes an objective function in some distance metric space. In terms of the selection of the objective function, these works can be divided into following categories where the optimal candidate location (i) MaxInf [5, 7–9, 21, 22]: maximizes the total weight of objects that are closer to it than to any other facility, i.e., has maximum influence, (ii) MinDist [3, 4, 14, 22, 24, 26]: minimizes the average distance of the objects from its nearest facilities, (iii) MinMax [2, 22]: minimizes the maximum distance of the objects from its nearest facilities, (iv) MaxSum [12]: maximizes the number of objects which will have the new facility as the nearest one. All these objective functions have been defined either in Manhattan [5, 26], Euclidean [3, 8, 9, 14, 21] or Road Network [2, 4, 14, 22, 24] space. These works can also be categorized based on the continuity of the solution space: (i) Discrete [3, 4, 7–9, 14, 21], and (ii) Continuous [2, 5, 12, 22, 24, 26]. They can be further categorized based on the number of optimal locations they return. Some works [4, 5, 14, 22, 26] find only a single optimal facility, whereas others [2, 3, 7–9, 12, 21, 24] find k optimal facilities. Table 1 gives an overview of the location selection queries in non-indoor setting.

Our work is similar to [2, 22] that also aim to optimize the MinMax objective function. However, both these works are intended exclusively for road network. Hence, the algorithms proposed in [2, 22] can not be adapted to indoor settings in a straightforward manner. We adapt [2] to find an optimal location indoor space and use it as a baseline to compare with our approach.

2.2 Indoor Route Planning Queries

An indoor trip planning query (iTPQ) [18] enables a user to visit at least one indoor point from each fixed set of categories with minimum indoor distance as the objective function. Category-aware multi-criteria route planning query (CAM) [17], a generalized version of iTPQ , lets a user choose the objective function, based on which an optimal route is calculated. Liu et al. [11] propose FPQ, a crowd-aware route planning query that returns the fastest route between two indoor points, and LCPQ [11] to return the route encountering the least number of objects. Feng et al. [10] study temporal-variation aware shortest path query (ITSPQ) that takes the opening and closing times of rooms and doors in an indoor space in contention while providing the solution. Indoor top-k keyword-aware routing query (IKRQ) [6] finds k routes with optimal ranking scores based on keyword relevance and distance cost. Salgado [15] proposes keyword-aware skyline route query (KSR) that takes the number of keywords visited in contention along with the route distance while calculating the solution routes. Different from the aforementioned route planning queries, given a moving source and a static destination point an indoor continuous detour query [16] continuously finds the nearest detour object that minimizes the indoor travel distance. Unlike these works that aim to find a route (a sequence of location) for a single client which optimizes some objective function satisfying a specified constraint, our work finds a single location for a set of clients which optimizes MinMax objective function.

2.3 Indoor Models and Indexes

Yang et al. [25] propose a door graph to model an indoor space, where each door represents a graph vertex. A weighted edge connects two vertices if the corresponding two doors belong to the same partition, where the edge weight is the indoor distance between the two doors. Graph traversal algorithms are then applied on the doors graph to compute and store the door to door distances in a hash table. Indoor distance-aware model [13] models the topology of an indoor space as an accessibility base graph where partitions represent vertices and doors represent directed edges between the two partitions they connect. It injects indoor distances into the base graph through a door to door and a door to partition indoor mapping function to support indoor shortest

Table 1: Existing Works in Non-Indoor Setting (D: Discrete, C: Continuous; M:Manhattan , E: Euclidean, RN: Road Network)

	Optimization Function				Solution Space		Distance Metric			|Query Answer|	
Reference	MaxInf	MinDist	MinMax	MaxSum	D	C	M	E	RN	1	k
[2]			✓	✓		✓			✓	✓	✓
[22]	✓	✓	✓			✓			✓	✓	
[4]		✓			✓				✓	✓	
[7]	✓				✓						✓
[21]	✓				✓			✓			✓
[5]	✓					✓	✓			✓	
[24]		✓				✓			✓	✓	✓
[26]		✓				✓	✓			✓	
[12]				✓		✓			✓		✓
[14]		✓			✓			✓		✓	
[8]	✓				✓			✓		✓	✓
[9]	✓				✓			✓			✓
[3]		✓			✓			✓			✓

distance and path queries. Indoor distance-aware index [13], an indexing framework for indoor spaces stores the global door-to-door distances and their ordering on top of this distance-aware model for further efficient computation of these indoor queries over the distance-aware model.

Xie et al. propose a composite indoor index [23], which is a combination of these three layers: i) *geometric layer:* it uses a R*- tree [1] for indexing indoor partitions, ii) *topological layer:* it stores the information about what two indoor partitions are connected and by which doors they are connected, and iii) *object layer:* it stores the information about which partition of the indoor space each object is in. Finally, Shao et al. [19, 20] propose two tree based indexes, Indoor-Partitioning tree (IP-Tree) and vivid IP-Tree (VIP-Tree) for indexing indoor space. Both these tree indexes are built bottom-up, and the index nodes store additional information for efficiently calculating the indoor distances. We discuss the VIP-Tree in detail in Section 3.

3 PROBLEM AND PRELIMINARIES

Research Problem (The Indoor Facility Location Selection (IFLS) Query). Given a client set C, an existing facility set F_e and a candidate location set F_n, an indoor facility location selection query returns a candidate location $n \in F_n$ for a new facility to be established at, which minimizes the maximum distances of the clients to their nearest facilities.

Let $iDist(c, f)$ return the indoor distance between a client c and a facility f, and $NN(c, F)$ return c's nearest neighbor from a set of indoor facilities F. Assuming A is the query answer, the indoor facility location query can then be formalized as follows.

$$A = \arg\min_{n \in F_n}(\max_{c \in C} iDist(c, NN(c, F_e \cup n)))$$

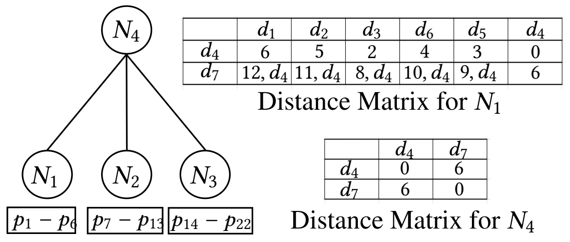
To clarify, our problem setting considers an existing facility or a candidate location as a partition of the indoor space.

Preliminaries: VIP Tree [19]. Vivid indoor partitioning (VIP) tree is the state-of-the-art index that partitions the indoor space while taking its topological properties such as multiple levels into account. First, it combines adjacent indoor partitions to create multiple leaf nodes and then builds the tree on top of the leaf nodes by combining the adjacent ones to generate the immediate-level non-leaf nodes. It keeps on doing so until it converges to a single root node. Figure 2 showcases the VIP tree built for the indoor space showed in Figure 1. Here, the nearby indoor partitions p_1 to p_6, p_7 to p_{13}, p_{14} to p_{22} are combined to generate the leaf nodes N_1, N_2 and N_3, respectively. Finally, all three leaf nodes are combined to generate the root node N_4.

To facilitate efficient indoor distance calculation, VIP-tree nodes store additional information in the form of distance matrices. The VIP-tree leaf nodes store the distances between all the doors and access doors of the node. On the other hand, the VIP-tree non-leaf nodes store the distances between the access doors of all its children. Access doors refer to those doors of a node which connect the node itself to the outside world, i.e., any path entering or leaving the node must go through these doors. Let d_1, d_2, d_3, d_5 and d_6 represent the doors of partitions p_1, p_2, p_3, p_5, and p_6, respectively. Moreover, let d_4 be the door connecting partition p_4 to p_7, and d_7 be the door connecting p_7 to p_{22}. Then, any path entering node N_1 have to use d_4, thus making d_4 the access door of leaf node N_1.

Aside from the distances, both the leaf and non-leaf nodes also store the first-hop door on the shortest path between two doors of the distance matrix. In addition, VIP-tree leaf nodes store the distance between all its doors and all the access doors of its ancestor nodes along with the first-hop door information.

Figure 2 illustrates the distance matrix of leaf node N_1 and root node N_4. In N_1, the corresponding entry between door d_1 and d_7 is $(12, d_4)$, i.e., the shortest distance between the two doors is 12 and d_4 is the first-hop door in the shortest path from d_1 to d_7.



	d_1	d_2	d_3	d_6	d_5	d_4
d_4	6	5	2	4	3	0
d_7	$12, d_4$	$11, d_4$	$8, d_4$	$10, d_4$	$9, d_4$	6

	d_4	d_7
d_4	0	6
d_7	6	0

Figure 2: VIP-tree for the indoor venue in Figure 1.

4 MODIFIED MINMAX ALGORITHM

MinMax algorithm [2] is the most efficient solution in the literature for identifying the optimal candidate location for a new facility in road networks. MinMax algorithm takes the road network as a graph input and selects the new facility location along one of its edges that minimizes the maximum road network distance of the clients from their nearest facilities.

Though indoor space has also been modeled as a graph in literature [13] where each indoor partition represents a vertex and each door represents an edge between two connected partitions, it performs not so well as the other indoor models and indexes [19] discussed in Section 2.3. On top of that, contrary to road networks where the movements of clients are restricted on roads, the clients can move freely inside a partition and the movements between partitions are restricted through doors and stairs. These make it non-trivial to apply the techniques proposed in [13] to model the indoor space, especially when the number of clients is large which is the case for indoor facility location selection queries.

Hence, we modify the MinMax algorithm to find the new facility location from a given set of indoor candidate locations. The key difference between the original MinMax algorithm and our modified version are as follows: (i) MinMax algorithm refines the continuous space instead of a candidate set, (ii) MinMax algorithm uses an efficient technique for finding the nearest existing neighbors for all clients in road networks, which we cannot use. Rather, we model the indoor space as a VIP-tree and apply relevant indoor distance calculation techniques [19] instead of the Dijkstra-like expansion techniques proposed in the original MinMax algorithm.

Algorithm 1 IFLS_Modified_MinMax(F_e, F_n, C)

```
1: $i \leftarrow 1$
2: $L_s \leftarrow Find\_NEF(C, F_e)$
3: $CA_i \leftarrow Find\_CA\_client(L_s^i.c, L_s^i.dist, F_n)$
4: $CA \leftarrow CA_i$
5: while $i < n$ and $|CA| > 1$ do
6:     $CA_{prev} \leftarrow CA$
7:     $i \leftarrow i + 1$
8:     $CA_i \leftarrow Find\_CA\_client(L_s^i.c, L_s^i.dist, CA)$
9:     $CA \leftarrow CA \cap CA_i$
10:    $j \leftarrow 1$
11:    while $j < i$ and $CA \neq \emptyset$ do
12:        $CA \leftarrow Refine\_CA(L_s^i.dist, CA[j], CA)$
13:        $j \leftarrow j + 1$
14:    end while
15: end while
16: $A \leftarrow Find\_Ans(CA, CA_{prev}, L_s, i)$
17: return $A$
```

Algorithm 1 shows the pseudocode of the modified MinMax algorithm. Its main steps are as follows:

(1) *Nearest existing facility computation:* The algorithm finds the nearest existing facility from each client in C from the existing facility location set F_e and stores them in a sorted list L_s (Line 2), where i^{th} entry of L_s includes a client c, its nearest existing facility e, the distance $dist$ between c and e. The entries of L_s are sorted based on the distances of the clients from their nearest existing facilities.

(2) *Candidate answer set CA generation:* The algorithm considers the first client with the maximum distance from her nearest existing facility and finds a candidate answer set CA from the candidate location set F_n. To elaborate, it finds the candidate locations in CA_1 for which $L_s^1.c$ has distances smaller than its distance from the nearest existing facility (Line 3) and assigns CA_1 to the candidate answer set CA (Line 4).

(3) *Candidate answer set CA refinement:* If the candidate answer set CA has more than one location, the algorithm considers other clients one by one in descending order of their distances from their nearest existing facilities. For each client, the algorithm refines CA by applying two pruning techniques:

   (a) The algorithm prunes the locations from CA whose distances from the client are larger than the client's distance from her nearest existing facility.

   (b) The algorithm prunes the locations from CA whose distances from any of the previously considered clients are larger than the current client's distance from her nearest existing facility.

   In each iteration i in the loop (Lines 5–15), the algorithm considers the next client from L_s, finds its CA_i (Line 8) using $Find\_CA\_client$ and refines CA (Lines 9-16).

   Function $Find\_CA\_client$ computes the distances of each candidate location in CA from $L_s^i.c$ and include those in CA_i whose distances are less than $L_s^i.dist$. Line 9 refines CA with respect to CA_i (pruning technique 1). Starting from the first client $L_s^i.c$, the inner loop (Lines 11–14) considers first $i-1$ clients in L_s one by one, and for each considered client $L_s^j.c$

   Function $Refine\_CA$ prunes a candidate facility location from CA if the candidate facility's distance from $L_s^j.c$ exceeds $L_s^i.dist$ (pruning technique 2). If the candidate answer set CA becomes empty for a client, then the algorithm stops refining CA for other remaining clients (Line 11).

(4) *Termination of the refinement:* The refinement terminates when either all clients are considered or there is one or no candidate location in CA (Line 5).

(5) *Finding the answer:* Finally, the algorithm (Line 16) finds the answer using $Find\_Ans$. If CA is empty, $Find\_Ans$ first assigns CA_{prev} to CA. If CA includes one candidate location, it becomes the answer A. If CA includes more than one candidate location, the function selects the one with the smallest maximum distance from the considered clients.

We index F_e and F_n separately using the most efficient indexing technique for the indoor space, VIP-tree, and use the VIP-tree based nearest neighbor and shortest distance computation algorithms [19] for indoor venues for implementing the steps mentioned above. Note that F_e is indexed once offline and the query parameter F_n is indexed during the query processing time.

We explain the steps of the modified MinMax algorithm using the example shown in Figure 1. The algorithm starts by sorting the clients in C based on their distances from their respective nearest existing facility in F_e in the following order.

$$L_s = \{(c_{25}, e_3, 8.90) > (c_{26}, e_3, 8.85) > (c_{23}, e_3, 8.70) > \\ (c_{24}, e_3, 8.68) > \ldots > (c_{58}, e_4, 0) > (c_{59}, e_4, 0)\}$$

Since client c_{25} has the maximum distance from her nearest existing facility e_3, $L_s^1.c = c_{25}, L_s^1.e = e_3, L_s^1.dist = 8.90$. The algorithm calculates the candidate answer set CA by considering $L_s^1.c$, which is c_{25}. CA_1 represents the candidate locations from F_n that have shorter distance than the distance between c_{25} and her nearest existing facility (i.e., $L_s^1.dist = 8.90$). The candidate answer set CA is generated from CA_1:

$$CA = CA_1 = \{n_5, n_9, n_{10}, n_{11}, n_{12}, n_{13}\}$$

Since, $|CA| = 6 > 1$, the modified MinMax algorithm picks the next client, $L_s^2.c = c_{26}$ off the sorted list and refines CA considering client $L_s^2.c = c_{26}$ based on the pruning strategies mentioned in Step 3 of modified MinMax algorithm. First, the algorithm calculates CA_2 by finding the candidate facility locations that have shorter distance than $L_s^2.c = c_{26}$'s nearest existing facility, $L_s^2.e = e_3$ and updates CA by picking only those locations common to both CA and CA_2, thus pruning the locations from CA whose distances are greater than client $L_s^2.c$'s distance to her nearest existing facility (pruning strategy 3a).

$$CA_2 = \{n_5, n_9, n_{10}, n_{11}, n_{12}, n_{13}\}$$
$$CA = CA \cap CA_2 = \{n_5, n_9, n_{10}, n_{11}, n_{12}, n_{13}\}$$

Then, the algorithm considers each of the previously considered clients, $L_s^1.c = c_{25}$ in this case and prunes those candidate locations (CA_1') from CA, which have larger distances than the current considered client $L_s^2.c = c_{26}$'s distance to her nearest existing facility, $L_s^2 = e_3$ (i.e., 8.85). (pruning strategy 3b).

$$CA_1' = \{n_i \in CA_1 : iDist(L_s^1.c, n_i) > L_s^2.dist\} = \emptyset$$
$$CA = CA \setminus CA_1' = \{n_5, n_9, n_{10}, n_{11}, n_{12}, n_{13}\}$$

Similarly, the modified MinMax algorithm picks the clients, $L_s^{3,4,5}.c = \{c_{23}, c_{24}, c_{28}\}$ one by one off the sorted list and refines the candidate location set CA using the two pruning strategies above. Only the calculation for client $L_s^5.c = c_{28}$ is shown below.

$$CA_5 = \{n_5, n_9, n_{10}, n_{11}, n_{12}, n_{13}\}$$
$$CA = CA \cap CA_5 = \{n_5, n_9, n_{10}, n_{11}, n_{12}, n_{13}\}$$
$$CA_1' = \{n_i \in CA_1' : iDist(L_s^1.c, n_i) > L_s^5.dist\} = \{n_{13}\}$$
$$CA_2' = \{n_i \in CA_2' : iDist(L_s^2.c, n_i) > L_s^5.dist\} = \{n_{13}\}$$
$$CA_3' = \{n_i \in CA_3' : iDist(L_s^3.c, n_i) > L_s^5.dist\} = \{n_{13}\}$$
$$CA_4' = \{n_i \in CA_4' : iDist(L_s^4.c, n_i) > L_s^5.dist\} = \{n_{13}\}$$

$$\therefore CA = CA \setminus (CA_1' \cup CA_2' \cup CA_3' \cup CA_4') = \{n_5, n_9, n_{10}, n_{11}, n_{12}\}$$

It goes on until the algorithm finds some client $L_s^{37}.c = c_3$ for which the candidate answer set can not be refined anymore, but only after the algorithm has considered the following clients, $L_s^{1,\dots,36}.c = \{c_{25}, c_{26}, c_{23}, c_{24}, c_{13}, c_{12}, c_{11}, c_{27}, c_{30}, c_{31}, c_{22}, c_{34}, c_7, c_{32}, c_5, c_{33}, c_{21}, c_{40}, c_{38}, c_6, c_{35}, c_{14}, c_{43}, c_{36}, c_{39}, c_{41}, c_{15}, c_{37}, c_{47}, c_{44}, c_{46}, c_{20}, c_{60}, c_4, c_{49}, c_{10}\}$ in order.

$$CA = \{n_5, n_{10}\}, CA_{37} = \{n_3\}; CA = CA \cap CA_{37} = \emptyset$$

Hence, the final candidate answer set, $CA = \{n_5, n_{10}\}$. As $|CA| > 1$, the modified MinMax algorithm selects the candidate location from CA which minimizes the maximum distance from the already considered clients $L_s^{1,\dots,36}.c$. Hence,

$$\mathrm{A} = \underset{n_j \in CA}{\arg\min} \left(\max_{c_i \in L_s^{1,\dots,36}.c} iDist(c_i, n_j) \right) = n_5$$

5 AN EFFICIENT APPROACH

The Modified MinMax algorithm considers each client separately while calculating their nearest existing facilities. Given the large number of clients in an indoor space, this significantly increases the number of required indoor distance computation and limits the performance of the algorithm. We develop an efficient approach for finding the optimal location to set up a new facility that minimizes the maximum distances of the clients from their nearest facilities. The key idea behind the efficiency of our approach is the refinement of the search space and reduction in the number of indoor distance computations. Moreover, rather than considering each client separately we group the nearby clients, which also reduces the indoor distance computation overhead. In addition, our efficient technique to incrementally find the nearest neighbours in the indoor space for all clients with a single search on the database further reduces the processing overhead.

5.1 Overview

In our approach, we index both existing and candidate facilities in $F_e \cup F_n$ using a single VIP-tree. The indexing of F_e in a VIP-tree is done once offline and then F_n is indexed as part of the same VIP-tree during the query processing. Our approach incrementally finds the nearest facilities for all clients in C with a single search on $F_e \cup F_n$. We develop an efficient *bottom up* VIP-tree traversal technique to incrementally find the nearest facilities for all clients with a single search. The traditional VIP-tree based nearest neighbour algorithm [19] starts the search from the VIP tree root and traverses the tree in a top down manner using a priority queue. Extending it for finding the nearest facilities for all clients would result in traversing majority of the leaf nodes because clients are distributed across different leaf nodes of the VIP tree. To reduce the query processing overhead of the top down search, we opted for the bottom up traversal of the VIP-tree. Initially, for each client we insert the leaf node that contains the client's partition into the priority queue and dequeue the queue. If the dequeued entry is a VIP-tree node, the algorithm enqueues its parent and child nodes into the priority queue if they have not been previously enqueued for that client, and the process carries on. Refer to Section 5.3 for a detailed explanation.

On top of that, we devise an efficient pruning technique to prune the clients whose distance from their nearest existing facilities can not be improved by a candidate location. The search proceeds while pruning clients from C based on the pruning technique proposed in Lemma 5.1. It stops when at least the closest facility with respect to every client has been identified. At this stage, the algorithm checks whether the answer is already determined by examining C, since C may change due to the pruning of clients. If C becomes empty while pruning the clients in the search process, then no answer exists for the query. If the remaining clients in C have a common retrieved nearest facility that is included in F_n, this location is returned as the answer. Otherwise, if the remaining clients in C have no common retrieved nearest facility in F_n, the algorithm again resumes the incremental nearest facility search until C becomes empty or a common retrieved nearest facility in F_n for all remaining clients in C has been identified.

The notations used in our algorithms are summarized below:

- Variable p represents any partition and variable $c_i.p$ represents the partition where client c_i is located.
- Variable N represents a VIP-tree node (i.e., the minimum bounding box of the partitions represented by the VIP-tree node).
- Variable $\mathcal{I}$ represents an indoor entity which can be an indoor partition or a VIP tree node.
- List L_i represents the list of retrieved facilities (existing and/or candidate) for client c_i, and the locations are sorted based on their shortest indoor distances from c_i.
- L represents the set of L_is for all clients in C.
- Variable $C'[p]$ represent the set of clients that have not been pruned and are located in partition p.
- C' is the set of $C'[p]$s for all partitions, i.e., $\bigcup_{e \in C'} \forall e = C$.
- Q is a priority queue where entries are ordered based on the shortest indoor distance between a partition p and an indoor entity $\mathcal{I}$ included in an entry.
- Function $iMinD(p, \mathcal{I})$ returns the shortest indoor distance between a partition and an indoor entity $\mathcal{I}$.
- Function $iDist(c, p)$ returns the shortest indoor distance between a client c and an indoor partition p.
- Variable G_d represents the shortest distance between a partition p and an indoor entity $\mathcal{I}$ of the last dequeued entry.
- Variable d_{low} represents the lower bound of the maximum shortest distance of the clients from their nearest facilities after the query answer A is identified.

5.2 Search Space Refinement

Our efficient approach considers F_e and F_n at the same time, which allows us to derive a *global distance* (G_d) for the refinement of the search space. Here, the global distance refers to the maximum distance between the clients and facilities (existing or candidate) that have already been retrieved from the database. We use this global distance to refine the search space in two ways: by pruning the facilities and the clients. The global distance guarantees the following: a VIP-tree node or a partition whose shortest distance from any client in C is less than or equal to the global distance have been retrieved from the database. Unlike modified MinMax algorithm, our approach does not need to find the nearest existing facility from F_e for a client, if the distance of the client from the nearest existing facility is greater than the global distance.

We prune a client if it is guaranteed that a candidate facility cannot improve the client's distance from her nearest existing facility. Once a client is pruned, our approach does not need to (i) retrieve any candidate facility for the client from the database and (ii) compute the distance of the client from the candidate facilities that are retrieved from the database for other clients. The following lemma shows the condition of pruning a client:

LEMMA 5.1. *Let $iDist(c, e)$ be the indoor distance between a client c and an existing facility $e \in F_e$, and G_d be a global distance that tracks the maximum distance between the clients and facilities (existing or candidate) that have already been retrieved from the database. A client c can be pruned if $G_d \geq iDist(c, e)$.*

PROOF. Let d_A represent the maximum shortest distance of the clients from their nearest facilities after the IFLS query answer A is found, which is mathematically defined as follows:

$$d_A = \max_{c \in C} iDist(c, NN(c, F_e \cup A))$$

If there is an existing facility e in the list of retrieved facilities for a client c, where $d_A \geq iDist(c, e)$, then the client can be pruned because the query answer A can not improve $iDist(c, e)$. At any time, $G_d \geq d_A$ (from definition). Thus, $G_d \geq iDist(c, e)$. □

5.3 Algorithm

Algorithm 2 shows the pseudocode of our efficient approach to solve the IFLS query. The inputs to the algorithm are the set of existing facilities F_e, the set of candidate locations F_n, and the set of clients C. The algorithm returns the optimal candidate location of the new facility A that minimizes the maximum of the distances of the clients from their nearest facility.

The algorithm starts with checking whether a client is located inside a facility. Specifically, if the partition where a client c_i is located is an existing or a candidate facility, i.e., included in F_e or F_n (Line 2), then the algorithm adds $c_i.p$ to the client's list L_i of retrieved facilities from the database (Lines 1–5). If the condition in Line 2 is true, then $c_i.p$ is the nearest facility of c_i with $iDist(c_i, c_i.p)$ equal to 0. At this stage, if a partition of any client c_i is an existing facility then according to Lemma 5.1, c_i can be safely pruned because A cannot further improve the c_i's distance from its nearest facility, which is already 0. Thus, in Line 6, the algorithm prunes such clients for whom one of the retrieved facilities is an existing facility from C and C', and their L_i from L using function *prune*. The parameters of this function are L, C', C, and d_{low}, where d_{low} is set to 0.

Next in Line 7, the algorithm checks whether the closest facility (existing or candidate) for all clients have been identified using Function $checkList$. The parameters of the function are L, C and G_d, where G_d is now set to 0 as the algorithm has only retrieved facilities for a client which are located in the same partition as the client. For each list in L, the function only considers those facilities in the list whose distances are smaller than or equal to G_d and sets the flag $isFirst$ to 1, if there is at least one facility in every list of L. Otherwise, $isFirst$ is set to 0.

If $isFirst$ is 1, the algorithm (Line 9) checks if the answer A is already found using Function $checkAnswer$. The parameters of the function are L, C and G_d. Flag $isAns$ is set to 1, if A is found; 0 otherwise. Function $checkAnswer$ works as follows:

- If C is empty then no answer exists for the query.
- If the remaining clients in C have a common retrieved nearest candidate facility in every list of L for the remaining clients and this common facility has distances less than or equal to G_d, this facility is returned as the answer.

If the answer is not found, the algorithm executes $exploreTree$ function, which is shown using Algorithm 3. Algorithm 3 uses a priority queue Q to explore the VIP-tree, where facilities in both F_e and F_n are indexed. Each entry of Q consists of a partition p where at least one client is located, $\mathcal{I}$, and $iMinD(p, \mathcal{I})$. The entries in Q are ordered based on $iMinD(p, \mathcal{I})$s.

For each partition p with at least one client in it, Algorithm 3 initially enqueues p, its parent N (the VIP tree leaf node that p is located in) and $iMinD(p, N)$. Next the algorithm iterates until the query answer is identified (Lines 7–36). In each iteration, the algorithm dequeues an entry and updates G_d (Lines 8–9). If the dequeued $\mathcal{I}$ is a facility, the algorithm adds $\mathcal{I}$ to L_i for each client c_i located in the dequeued partition p (Lines 10–13). On the other hand, if $\mathcal{I}$ is a VIP tree node, the algorithm enqueues entries to Q by considering parent and child nodes of $\mathcal{I}$ (Lines 14–22). While enqueueing, the algorithm ensures that the node has not been visited before (i.e., $N' \leftarrow parent(\mathcal{I})$ in Line 15 or $ch \neq p$ in Line 19). To clarify, given the fact that $\mathcal{I}$ represents either an indoor partition or a VIP tree node, N' will always be a VIP tree non-leaf node, whereas ch can be either a partition or a VIP tree node.

If $isFirst$ flag is false, in Line 24, the algorithm checks again whether the closest facility (existing or candidate) for all clients have been identified at this stage using Function $checkList$. If $isFirst$ flag is still false, the algorithm prunes the clients from C' and C whose distances from an existing facility is smaller than or equal to G_d (Line 27) according to Lemma 5.1, because it is guaranteed that the minimum distance of the clients from their nearest facilities after the query answer is identified will not be smaller than G_d as the closest facility for all clients have not yet been identified. The algorithm updates d_{low} to G_d (Line 28).

On the other hand, if $isFirst$ is true, the algorithm iterates in a loop (Lines 31–35). In each iteration, the algorithm updates d_{low} using Function $increaseDist$ as follows. Function $increaseDist(L, d)$ sets d_{low} to a distance of a facility, where the the facility's distance from any client is greater than current d_{low} and smaller than or equal to G_d, and the facility's distance is the smallest among all facilities that satisfy the above two conditions.

The intuition behind this step is as follows. Since, we consider a partition instead of an exact location of a client while enqueueing an entry to Q, Q is sorted based on $iMinD$ instead of $iDist$. As a result, when a facility is added to a list of a client in Line 12, it may happen that the client's actual distance $iDist$ from the facility becomes greater than G_d. This facility is not eligible to consider for the client while computing the answer in the current iteration as there may be other facilities that have smaller distances from the

Algorithm 2 IFLS_EA(F_e, F_n, C)

```
 1: for each $c_i \in C$ do
 2:     if $c_i.p \in (F_e \cup F_n)$ then
 3:         $L_i \leftarrow addFacility(c_i, c_i.p, 0)$
 4:     end if
 5: end for
 6: $prune(L, C', C, 0)$
 7: $isFirst \leftarrow checkList(L, C, 0)$
 8: if $isFirst$ = true then
 9:     $A, isAns \leftarrow checkAnswer(L, C, 0)$
10: end if
11: if $isAns$ = false then
12:     $A, isAns \leftarrow exploreTree(F_e, F_n, C, C', L, isFirst)$
13: end if
14: return $A$
```

Algorithm 3 exploreTree($F_e, F_n, C, C', L, isFirst$)

```
 1: $Q \leftarrow \emptyset$
 2: $end, d_{low} \leftarrow 0$
 3: for each partition $p$ with $|C'[p]| > 0$ do
 4:     $N \leftarrow parent(p)$
 5:     $Enqueue(Q, p, N, iMinD(p, N))$
 6: end for
 7: while $!isAns$ do
 8:     $p, \mathcal{I}, iMinD(p, \mathcal{I}) \leftarrow Dequeue(Q)$
 9:     $G_d \leftarrow iMinD(p, \mathcal{I})$
10:     if ($\mathcal{I}$ is a facility and $|C[p]| > 0$) then
11:         for each client $c_i \in C'[p]$ do
12:             $L_i \leftarrow addFacility(c_i, \mathcal{I}, iDist(c_i, \mathcal{I}))$
13:         end for
14:     else if ($\mathcal{I}$ is not a facility) then
15:         if $parent(\mathcal{I}) \neq ancestor(p)$ then
16:             $N' \leftarrow parent(\mathcal{I})$
17:             $Enqueue(Q, p, N', iMinD(p, N'))$
18:         end if
19:         for each child $ch$ of $\mathcal{I}$, with $ch \neq p$ do
20:             $Enqueue(Q, p, ch, iMinD(p, ch))$
21:         end for
22:     end if
23:     if ($isFirst$ = false) then
24:         $isFirst \leftarrow checkList(L, C, G_d)$
25:     end if
26:     if ($isFirst$ = false) then
27:         $prune(L, C', C, G_d)$
28:         $d_{low} \leftarrow G_d$
29:     else
30:         $end \leftarrow 0$
31:         while $isAns$ = false do
32:             $d_{low}, end \leftarrow increaseDist(L, d_{low}, G_d)$
33:             if ($end = 0$) then
34:                 $prune(L, d_{low}, C', C)$
35:                 $A, isAns \leftarrow checkAnswer(L, C, d_{low})$
36:             end if
37:         end while
38:     end if
39: end while
40: return $A$
```

client and have not yet been retrieved from the database. However, in a later iteration, when G_d is increased (Line 9), more than one facility's distances from a client may become smaller than or equal to G_d. This is why the algorithm increases d_{low} in steps in a loop instead of directly assigning G_d to d_{low} and checks whether the answer is already identified for a newly considered facility.

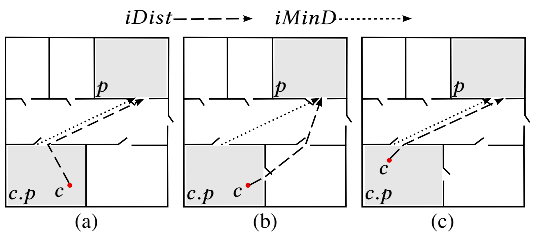


Figure 3: *iDist* **and** *iMinD* **calculation.**

If no such facility is found, the function sets *end* to 1, Otherwise, if $end = 0$, the algorithm prunes the clients that are eligible according to according to Lemma 5.1 using Function *prune* (Line 34) and checks whether the answer is identified using Function *checkAnswer* (Line 35).

5.3.1 iMinD and iDist Calculation. Here, we explain the steps of calculating $iMinD(p, \mathcal{I})$, the shortest distance between an arbitrary partition and an indoor entity, and $iDist(c, p)$, the shortest distance between a client and an arbitrary indoor partition.

$iMinD(p, \mathcal{I})$ calculation: We follow the same approach of calculating the shortest distance between two arbitrary indoor points proposed in [19] to calculate the shortest distance between a partition p and an indoor entity $\mathcal{I}$. The only difference is the distance between an indoor point and the door of the partition it is located inside is either positive or 0, whereas the distance between an indoor partition and its doors is always 0.

$iDist(c, p)$ calculation: Let $c.p$ be the partition where client c is located and $D(c.p)$ be that partition's doors. Based on the number of doors of partition $c.p$, the shortest distance between an arbitrary client c and an arbitrary partition p can be calculated as follows.

Case 1 (Figure 3a). If $|D(c.p)| = 1$, we have to use $c.p$'s only door to calculate $iMinD(c.p, p)$ as client c also has to use that door to leave $c.p$. Hence, we use the already calculated $iMinD(c.p, p)$ to calculate $iDist(c, p)$ as follows, where $d \in D(c.p)$.

$$iDist(c, p) = iMinD(c.p, p) + iDist(c, d)$$

Case 2 (Figures 3b, 3c). When $|D(c.p)| > 1$, we get the information about the door client c uses to leave partition $c.p$ only after $iDist(c, p)$ is calculated. Hence, the already calculated $iMinD(c.p, p)$ can not be utilized in this case as the door used by client to leave partition $c.p$ may (Figure 3(b)) or may not (Figure 3(c)) be the same as the door used in calculating $iMinD(c.p, p)$. Thus, we follow the same approach of calculating the shortest distance between two arbitrary indoor points proposed in [19].

5.4 A Working Example

We explain the steps of our approach using the same example shown in Figure 1. Initially, the algorithm finds the partition a client is located in and includes it as one of the retrieved facilities for the client, if that partition is either an existing or a candidate facility (Line 2-5 in Algorithm 2). In Figure 1, client c_1 is located in partition $c1.p = p_2$, which is an existing facility (e_1). Hence, e_1 is the nearest facility of client c_1 with $iDist(c_1, e_1) = 0$ and $(e_1, 0)$ is inserted into L_1. On the other hand, client c_8 is located in partition $c_8.p = p_4$, which is neither an existing facility nor a candidate location. Hence, no facility is retrieved for client c_8. Once this step is done for all clients in C, the algorithm moves on to the next step of pruning clients which have an existing facility in its L_i, i.e., the nearest facility of whom are an existing facility (Line 6 in Algorithm 2). The algorithm prunes clients $c_1, c_{17}, c_{18}, c_{52}, c_{58}$ and c_{59}, because the nearest facility of these clients are an existing facility: e_1, e_2, e_2, e_3, e_4, and e_4, respectively.

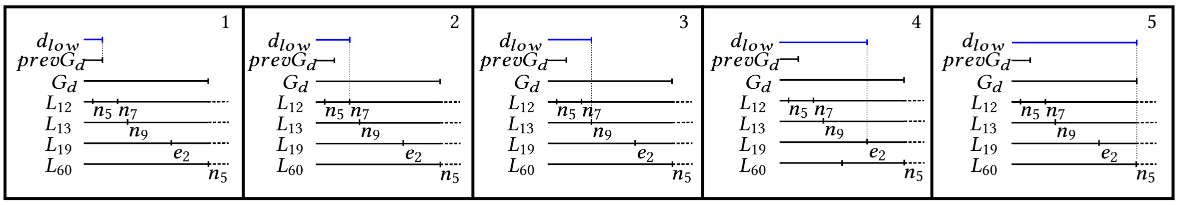


Figure 4: *increaseDist* **calculation**

Consequently, the list of the potential clients gets updated from C to $C = C - \{c_1, c_{17}, c_{18}, c_{52}, c_{58}, c_{59}\}$. C' and $C'[p]$s are also updated accordingly.

Next, the algorithm calls function $checkList(L, C, 0)$, goes through the L_is' for each client in C and checks if the nearest neighbours within distance 0 have been retrieved for all these clients (Line 8 in Algorithm 2). For clients c_8, c_9 and c_{10}, the nearest neighbours within distance 0 have not been retrieved yet. So, flag $isFirst$ is set to false, and function $exploreTree$ (Algorithm 3) is invoked (Line 12-14 in Algorithm 2).

Algorithm 3 first populates Q with $(p, N, iMinD(p, N))$s, where p is a partition with $|C'[p]| \geq 1$ and N is the leaf node containing p. In Figure 1, $|C'[p_1]| = 0$. As a result, p_1 is skipped. However, $|C'[p_4]| = 2, parent(p_4) = N_1$ and $iMinD(p_4, N_1) = 0$. Hence, $(p_4, N_1, 0)$ is enqueued. Once Q is populated (Line 3-6), the algorithm iterates until $isAns = 1$.

During the first iteration the algorithm dequeues $(p_4, N_1, 0)$ and sets $G_d = 0$. N_1 is a leaf node of the VIP tree, so the algorithm enqueues N_1's parent node N_4 and children p_1, p_2, p_3, p_5, p_6 into Q. As $isFirst$ was set to false previously, the algorithm invokes $checkList(L, C, 0)$ to recalculate the flag. Since no facilities have been extracted in this iteration $isFirst$ remains false. As a result, the algorithm starts the next iteration after invoking function $prune(L, C', C, 0)$. During another iteration $(p_{15}, p_{10}, 7.23)$ gets dequeued, hence G_d is set to 7.23. Since partition p_{10} is a facility and $C'[p_{10}] = \{c_{26}, c_{27}\}$, the algorithm calculates $iDist(c_{26}, p_{10}), iDist(c_{27}, p_{10})$ and includes p_{10} to L_{26}, L_{27}. It then invokes $checkList(L, C, 7.23)$ and updates $isFirst = true$.

Since $isFirst$ has been updated to true, it calculates a sorted list of facilities $L_{(prevG_d, G_d]}$ the distance of which from a client is within the range $(prevG_d, G_d] = (6.80, 7.23]$. Here, $prevG_d$ = the G_d during the previous iteration and $L_{(prevG_d, G_d]} = \{(c_{12}, n_9, 6.98), (c_{13}, n_9, 7.10), (c_{19}, e_2, 7.18), (c_{60}, n_5, 7.23)\}$. The algorithm then starts iterating a loop. In the first iteration, it removes the first element $(c_{12}, n_9, 6.98)$ off the list and sets $d_{low} = 6.98$. Since n_9 is a candidate location, client c_5 is not pruned and function $checkAnswer(L, C, 6.98)$ is invoked to recalculate the flag $isAns$ = false. On the other hand, in the third iteration when it removes $(c_{19}, e_2, 7.18)$ client c_{19} is pruned because an existing facility e_2 has been retrieved for this client and C gets updated to $C = C \setminus c_{19}$. Figure 4 shows all the iterations of the loop after which $isAnsFound$ still remains false.

As a result, the algorithm moves on to the next iteration and follows the same procedure again. After *six* more of these iterations, the algorithm finally sets $isAnsFound$ to true when it finds a common candidate facility $A = n_5$ for clients in $C = \{c_5 - c_7, c_{10} - c_{15}, c_{21} - c_{25}, c_{21} - c_{25}, c_{28} - c_{35}, c_{38} - c_{41}, c_{42}, c_{43}\}$.

6 EXPERIMENTS

We present our experimental setup in Section 6.1. Then, we present the performance of our proposed approach and compare it with the modified MinMax-Alg for processing IFLS queries in Section 6.2.

6.1 Experiment Settings

We ran our experiments in an Intel (R) Core (TM) i5-3230M 2.60 GHz machine with 8 GB RAM running on Windows 10.

6.1.1 Datasets. We evaluate our model on four real world indoor venues:

- **Melbourne Central (MC)**[1]. A large shopping centre in Melbourne with 299 doors and 298 rooms spread over 7 levels.
- **Chadstone (CH)**[2]. The largest shopping centre in Australia with 678 doors and 679 rooms spread over 4 levels.
- **Copenhagen Airport (CPH)**[3]. The ground floor of Copenhagen Airport, spanning over 2000m × 600m with 76 rooms and 118 doors.
- **Menzies Building (MZB)**[4]. A landmark building at Clayton campus of Monash University, with 1344 rooms and 1375 doors spanning over 16 levels.

We use both real and synthetic setting to select/generate existing facilities, candidate facility locations.

- **Real Setting.** Client locations are generated using uniform and normal distribution. Existing facilities and candidate locations are selected from real data. We choose Melbourne Central (MC) venue to run our experiments in real setting. Generally, the partitions across an indoor venue can be categorized into different categories based on the service they provide. For example, the MC shopping center has different categories of partitions such as fashion & accessories, dining & entertainment, health & beauty, fresh food, banks & services and so on. For our experiments, we select the partitions of one of such categories e.g. dining & entertainment as existing facilities and the partitions of the remaining categories are selected as candidate locations.
- **Synthetic Setting.** We choose all 4 venues: MC, CH, CPH, MZB to run our experiments in synthetic setting. Unlike real setting, existing facilities and candidate locations are selected using uniform random distribution. Clients are generated using uniform, and normal distribution.

6.1.2 Parameter settings. We vary the following parameters: (i) the existing facility size $|F_e|$, (ii) the candidate location size $|F_n|$, (iii) the client size $|C|$, and (iv) the standard deviation of normal distribution σ for our experiments in synthetic setting. For real setting, we only vary (i) the client size $|C|$, and (ii) the standard deviation of Normal distribution σ. To select the set of existing facilities and candidate locations in real setting, we consider the following five categories of partitions: fashion & accessories (101), dining & entertainment (54), health & beauty (39), fresh food (19) and banks & services (14). When we consider fashion & accessories category, we select the 101 partitions of the mentioned category as our existing facility set and the remaining 190 partitions as the candidate location set. Table 2 shows the range used for each parameter, where the default values are marked in bold.

[1]https://www.melbournecentral.com.au/centre-info/centre-map
[2]https://www.chadstone.com.au/directions/centre-map/
[3]https://www.cph.dk/en/practical
[4]https://www.monash.edu/virtual-tours/menzies-building

Table 2: Parameter settings for Indoor Facility Location Selection Query.

Parameters	Range				
	Real setting	Synthetic setting			
Venue	MC	MC	CH	CPH	MZB
Existing facility size (F_e)	101, 54, **39**, 19, 14	[25, 125], Δ = 25	[50, 150], Δ = 25	[10, 30], Δ = 5	[100, 500], Δ = 100
Candidate location size (F_n)	190, 237, **252**, 272, 277	[100, 200], Δ = 25	[100, 500], Δ = 100	[25, 45], Δ = 5	[300, 700], Δ = 100
Client size (C)	1k, 5k, **10k**, 15k, 20k				
Normal Distribution:μ; σ	0; 0.125, 0.25, 0.5, **1**, 2				

In synthetic setting, the existing facility size and the candidate location size are varied within the range [a, b] with the defined Δ and the mean of these values are used as the default value. To observe the effect of a parameter in an experiment, the value of the parameter was varied within its range, and other parameters were set to their default values.

6.1.3 Evaluation Metric. We evaluate the performance of the efficient approach and modified MinMax algorithm on the following two metrics: i) processing time, and ii) memory cost. For each set of experiment, we ran 10 IFLS queries and then recorded the average processing time, and memory cost.

6.2 Performance Evaluation & Parameter Sensitivity

In this section, we present the experimental results by varying different parameters.

6.2.1 Effect of Client Size. Figures 5 and 7a show that the efficient approach comfortably outperforms the modified MinMax-Alg by one order of magnitude in terms of the query processing time with the increase of client size. For real setting (Figure 5), on average the efficient approach achieves a speedup of 8.28 times over the modified Min-Max algorithm, and reaches up to 24.96 times. For synthetic setting (Figure 7a), on average the efficient approach achieves a speedup of 14.23, 2.75 and 4.02 times over the modified Min-Max algorithm and reaches up to 22.26, 2.94, and 4.57 times for MC, CH and MZB dataset, respectively.

However, the efficient approach does worse than the Modified MinMax-Alg on CPH dataset due to its small size. The performance gain of the efficient approach stems from its pruning strategy which is dependent on the existing facility size. With a smaller existing facility size, less clients are pruned which results in the retrieval of more candidate locations for each client, thus increasing the indoor distance computation. On the other hand, for Modified MinMax-Alg, a smaller candidate location size ensures that the initial candidate answer set is far smaller, which makes it take less time to converge to the query answer.

As we increase the client size, the processing time of the efficient approach increases for both real and synthetic setting. Though the efficient approach groups the nearby clients in a same partition and enqueues these partitions in the priority queue (Q), it still considers each client separately while calculating the indoor distances. As the number of clients increase, the number of indoor distance calculation operations also increase. Besides, with the increase of client size the *checkList* and *prune* operations become more expensive since there are more clients to consider.As a result, the processing time increases.

The processing time of the modified MinMax-Alg increases sharply with respect to the client size $|C|$. Initially, the modified MinMax-Alg computes the nearest existing facilities from each client using VIP-tree based nearest neighbor and shortest distance computation algorithms. Hence, for each additional client the algorithm has to compute its nearest existing facility, which in turn increases the query processing time.

In real setting the existing facilities are generally clustered, whereas in synthetic setting the existing facilities are selected according to uniform random distribution. Despite the difference in the distribution of the existing facilities between real and synthetic setting, the minimal performance discrepancy between the efficient approach and the Modified Min-Max, as well as the similar steepness of both curves, indicate that the distribution of existing facilities has a negligible effect on IFLS query performance.

In terms of memory cost, the modified MinMax-Alg performs better than the efficient approach with the increasing client size. On an average the modified MinMax-Alg incurs 2.86 times less memory in real setting and 2.34 (MC), 3.88 (CH), 17.82(CPH), 7.29 (MZB) times less memory cost in synthetic setting than the efficient approach, respectively. However, the efficient approach reports a maximum memory cost of 41 MB for real setting and 468.94 MB for synthetic setting (MZB), which is negligible.

Both the efficient approach and the modified MinMax-Alg require more memory as the client size increases. However, compared to the real setting there is a steep increase in the required memory in synthetic setting with the increasing client size. This is due to the difference in the number of facilities in two settings: in synthetic setting, an additional client has to consider twice the number of facilities than in real setting.

6.2.2 Effect of Standard Deviation. For normal distribution, varying the standard deviation, σ means varying the degree of inclination for the data points to cluster at the central area of an indoor venue. A smaller σ leads to more dense data points at the center, whereas a larger σ leads to more dispersed data points. For both synthetic and real setting we only generate the client data using normal distribution.

For real setting (Figure 6(i)), on average the efficient approach achieves a speedup of 545.03 times over the modified Min-Max algorithm, and reaches up to 907.55 times in terms of the query processing time. In case of synthetic setting (Figures 6(ii)- 6(v)), on average the efficient approach achieves a speedup of 235.38 (MC) , 2.80 (CH), and 32.03 (MZB) times over the modified Min-Max algorithm and reaches up to 644.33 (MC), 3.17 (CH), and 39.09 (MZB) times.

The query processing time of the efficient approach increases with σ in real setting (Figure 6(i)). The existing facilities in real setting are clustered. As a result when clients become more dispersed with the increasing σ, the distance between an existing facility and a client rather increases. Hence, the number of pruned clients decrease which in turn increases the number of locations retrieved from the database. For the modified MinMax-Alg, this increase in distance increases the size of the candidate location set which requires more clients to be considered for the refinement step. However, the increase of the candidate location size is negligible, thus the processing time of the modified MinMax-Alg almost remains constant in real setting.

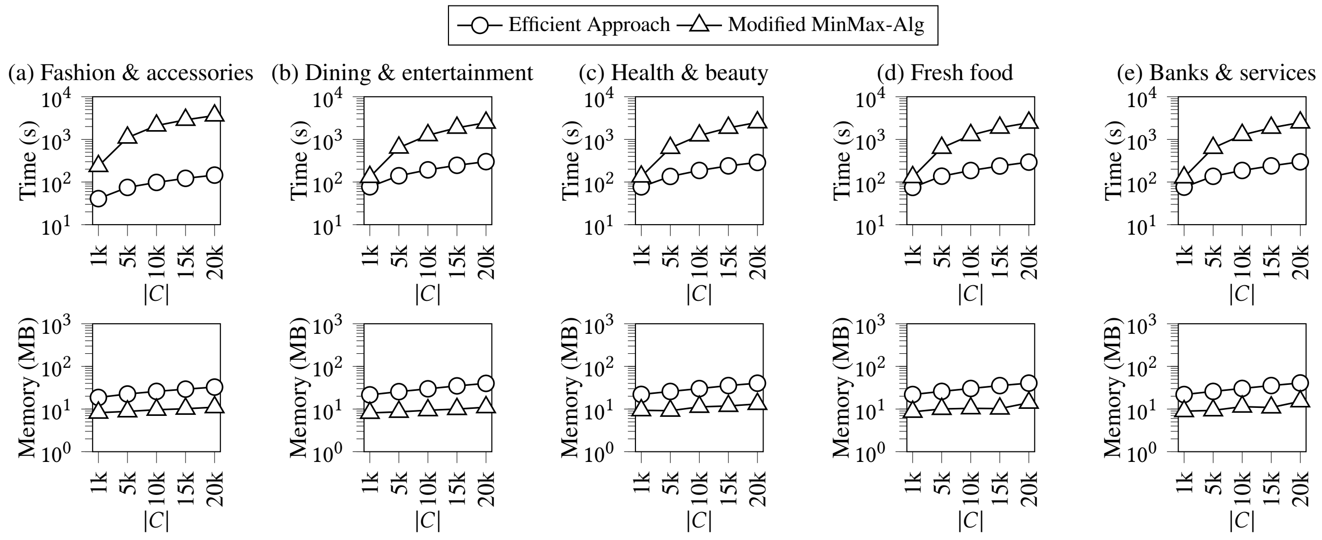


Figure 5: Effect of $|C|$ (Real Setting).

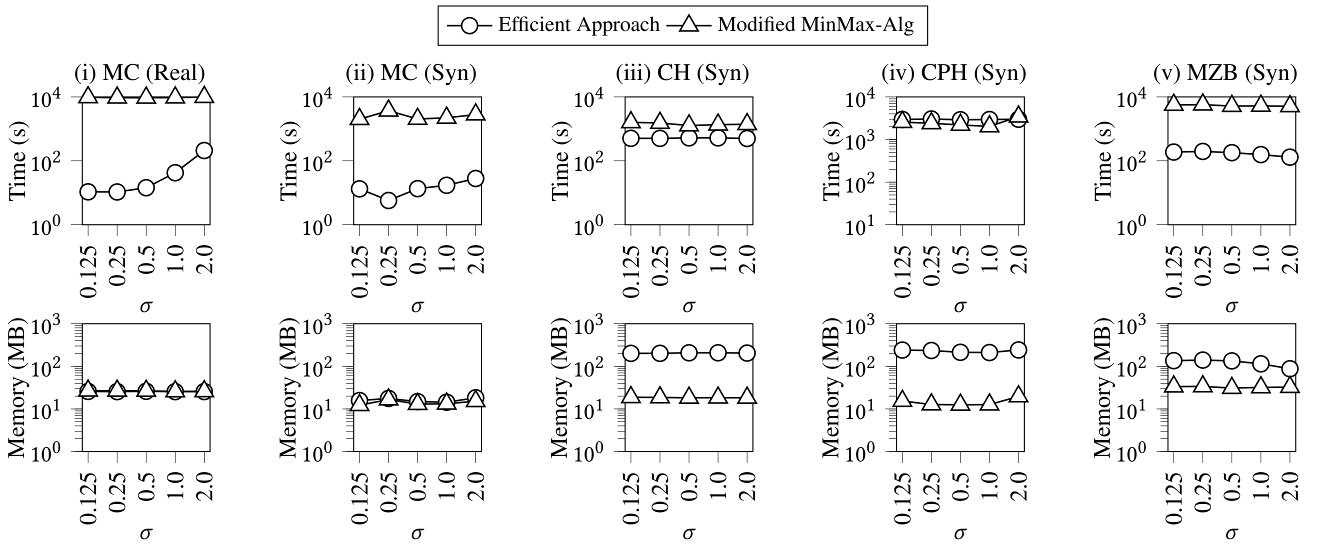


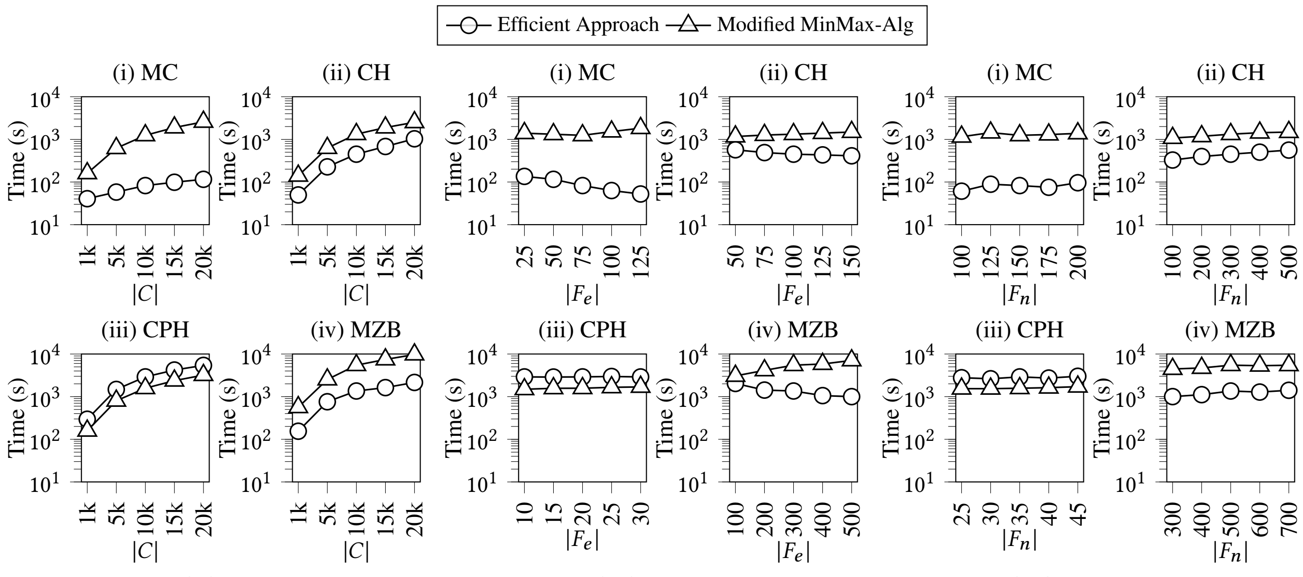
Figure 6: Effect of σ (Real & Synthetic Setting).

On the contrary, the processing time of both approaches decrease slightly with σ in synthetic setting (Figures 6(ii)- 6(v)). As σ increases, the clients become more dispersed while the existing facilities and candidate locations remain uniformly distributed. As the clients get more dispersed, the distance between an existing facility and a client decreases. For the efficient approach, this increases the number of pruned clients. For the modified MinMax-Alg, this decreases the size of the candidate location set. As the size of the candidate location set decreases, less number of clients are considered for refining the candidate location set. Hence, there is a decrease in query processing time.

In terms of memory cost, both the efficient and the modified MinMax-Alg approaches perform similarly. However, in synthetic setting the modified MinMax-Alg performs better than the efficient approach with the increasing standard deviation. On an average the modified MinMax-Alg incurs 11 times less memory cost than the efficient approach. It is also evident from Figure 6 that both approaches require almost a constant amount of memory with the increase of σ.

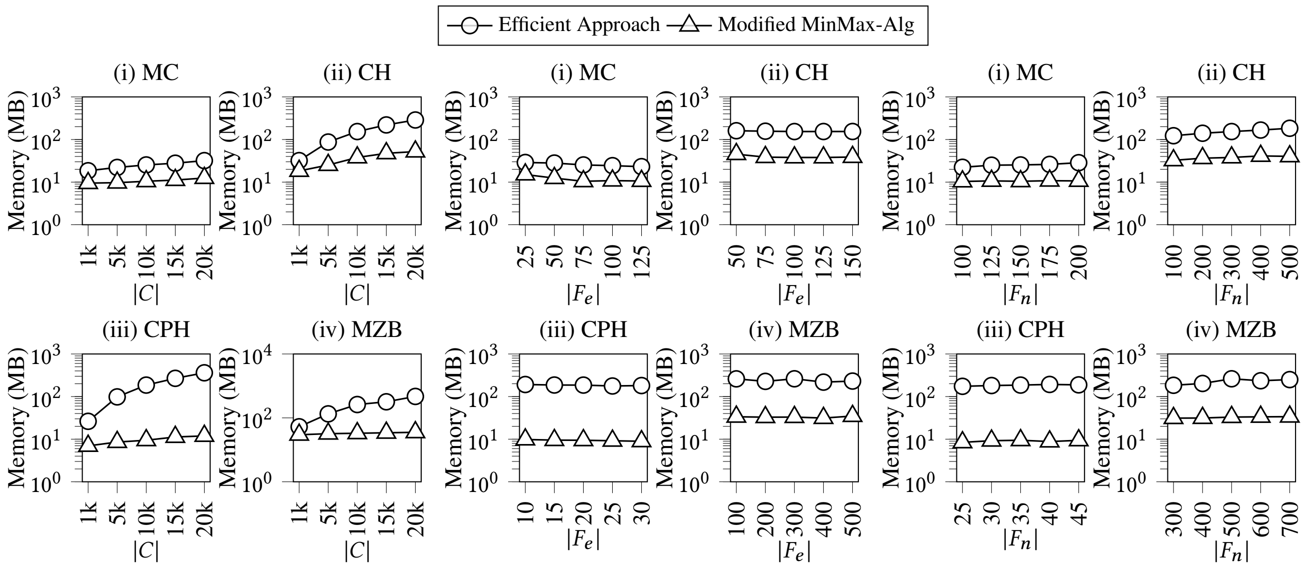
6.2.3 Effect of Existing Facility Size. Figure 7b shows that the efficient approach comfortably outperforms the modified MinMax-Alg with the increase of existing facility size, in terms of the query processing time. For synthetic setting, on average the efficient approach achieves a speedup of 19.18 (MC), 2.88 (CH) and 4.20 (MZB) times over the modified Min-Max algorithm and reaches up to 35.02 (MC), 3.61 (CH) and 7.07 (MZB) times.

The gain in speed of the efficient approach over the modified MinMax algorithm comes down to two reasons. First, the efficient approach groups the nearby clients in the same partition and enqueues the partitions in the priority queue Q as representatives of the clients in stead of the actual clients themselves. Since the number of partitions in an indoor venue is always bounded, it reduces the size of the priority queue which in turn decreases the number of probable dequeue operations. Besides, we utilize the case when a partition has a single door and use the already calculated indoor distances for that partition to calculate the indoor distances for the clients inside that partition. It reduces the computational overhead of calculating the indoor distances for all the clients that are in that partition from scratch. These factors combine together to reduce the query processing time compared to the modified MinMax-Alg, where each client is considered separately. Second, the efficient approach refines the search space by pruning clients for whom a candidate facility cannot improve its current distance to the nearest



(a) $|C|$ **vs Time** **(b)** $|F_e|$ **vs Time** **(c)** $|F_n|$ **vs Time**

Figure 7: Effect of $|C|$, $|F_e|$, $|F_n|$ **on query processing time (Synthetic Setting).**



(a) $|C|$ **vs Memory** **(b)** $|F_e|$ **vs Memory** **(c)** $|F_n|$ **vs Memory**

Figure 8: Effect of $|C|$, $|F_e|$, $|F_n|$ **on memory (Synthetic Setting).**

existing facility. This eliminates the cost of retrieving candidate facilities and calculating indoor distances for the pruned clients.

As the existing facility size increases, the processing time of the efficient approach decreases in synthetic setting (Figure 7a). With the increase of the existing facility size the distribution density of the existing facilities increases, which shortens the distance between each client and its nearest existing facility. As a result the number of clients which are pruned increases. This reduces the number of facilities or locations required to be retrieved from the database and the computational overhead of calculating the indoor distances for these locations.

On the contrary, the processing time of the modified MinMax-Alg increases with the increase of existing facility size. Initially, the modified MinMax-Alg computes the nearest existing facility incrementally for each client from a fixed set of existing facilities. As the existing facility size increases the computational overhead of calculating the indoor distances for each client increases, thus the increase in processing time.

In terms of memory cost, the modified MinMax-Alg performs better than the efficient approach as the existing facility size increases. On average the modified MinMax-Alg incurs 2.20, 3.94, 19.70 and 7.24 times less memory cost than the efficient approach for MC, CH, CPH and MZB dataset, respectively in synthetic setting. However, the efficient approach reports a maximum memory cost of 261.57 MB for synthetic setting (MZB), which is negligible considering the size of memory in current machines. This behavior is predictable considering the efficient approach considers all the clients at a time, which requires more memory to store all the clients and their retrieved facilities from the database. On the other hand, the modified MinMax-Alg considers each client separately and hence requires less memory. Figure 8b shows that both the efficient approach and the modified MinMax-Alg require less memory as the existing facility size increases which is down to the same reason discussed before.

6.2.4 Effect of candidate Location Size. Figure 7c shows that the efficient approach achieves a speedup of 16.35 (MC), 2.94 (CH) and 4.14 (MZB) times over the modified Min-Max algorithm and reaches up to 18.90 (MC), 3.29 (CH) and 4.44 (MZB) times with the increase of candidate location size for the same reason discussed in Section 6.2.3 in terms of the query processing time.

In synthetic setting, the processing time of the efficient approach increases steadily with the increase of candidate location size $|F_n|$ (Figure 7c). The increasing number of candidate location increases the distribution density of the candidate locations, i.e., the distance between each client and a candidate location decreases. This in turn increases the number of candidate locations which are retrieved for each client before an answer is found. This increased number of candidate locations increases the processing overhead of operations such as $checkList, prune, increaseDist$. That's why the query processing time of the efficient approach increases with candidate location size.

The processing time of the modified MinMax-Alg shows a similar trend. Once the modified MinMax-Alg completes step 1 (Nearest existing facility computation), it generates a candidate answer set with respect to the client with the maximum distance from its nearest existing facility. Candidate locations, which are closer to the client than it's nearest existing facility constitute the candidate set. The algorithm then keeps on refining the candidate set considering clients one by one in descending order of their distances from the existing nearest facilities until all the clients are considered or there is one or no facility in the candidate set at all. As the number of candidate locations increase, the distance between a client and its nearest candidate locations tend to decrease. Hence, the probable size of candidate set increases since an increasing number of candidate locations now have shorter distance than the client's nearest existing facility. The algorithm thus considers additional clients as the candidate location size increases which results in an increase in query processing time.

In terms of memory cost, on average the modified MinMax-Alg incurs 2.40, 4.05, 20.41 and 6.95 times less memory cost than the efficient approach for MC, CH, CPH, and MZB dataset, respectively in synthetic setting, where the efficient approach reports a maximum memory cost of 261.57 MB for synthetic setting. Figure 8c shows that both the efficient approach and the modified MinMax-Alg require more memory as the candidate location size increases. This is due to the increase in the number of candidate locations retrieved from the database for the reasons discussed above for the respective algorithms.

7 EXTENSION

Our efficient solution for the IFLS query can be extended for MinDist function that minimizes the average distance of the clients from their nearest facilities. The only difference is the way in which the candidate answer set is generated and $checkAnswer$ function works. Other than that algorithm workflow proposed in Section 5.3 and the client pruning technique proposed in Lemma 5.1 remain the same.

Initially, the candidate answer set contains all candidate locations. Gradually, the candidate answer set is refined until the query answer is found. Each element in the candidate answer set consists of the total distance of the clients from the corresponding candidate location and a flag indicating whether the total distance is the actual distance or a lower bound. Initially the total distance is initialized with ∞ and the flag indicates a lower bound. If a candidate location n in the current candidate answer set is included in the retrieved list of facilities for all the non-pruned clients, and the distance between n and all clients are less than the global distance, then the actual distance can be calculated for this candidate location. However, if for at least one of the non-pruned clients the distance between n and the client is unknown, then the actual distance can not be calculated. In this case, we use the global distance as the lower bound of the unknown distance between n and the client, and update the lower bound of the total distance for n. In both cases (total distance or its lower bound), for a pruned client if n is not in its retrieved list of facilities, we use the distance between the client and its closest existing facility. As facilities are retrieved from the database, the candidate locations that have greater distances (be it actual or not) than the candidate facility with the smallest actual total distance are pruned from the candidate answer set. Consequently, $checkAnswer$ requires the following modification:

- If C becomes empty, then the total distance for every candidate location included in the candidate answer set is known and the candidate location in the candidate answer that minimizes the total distance is returned as the answer.
- If C is non-empty and the total distance of at least one of the candidate locations of the candidate answer set is known and it is smaller than the distances of all other locations in the candidate answer set, then that location is returned as the query answer. If there are multiple such locations, all of them are returned.

Another variant of IFLS query, MaxSum maximizes the number of the clients who will have the new facility as the nearest one. Similar to MinDist function, our efficient solution can be modified for MaxSum function. For MaxSum function, the upper bound of the total count can be used to refine the candidate answer set.

8 CONCLUSION

We have introduced a new query service, namely *Indoor Facility Location Selection (IFLS)* query that finds the optimal location for placing a new facility in an indoor venue such that the maximum distance of all clients to their nearest facility is minimized. The IFLS query has a number of real-life applications in many indoor venues including hospitals, airports, universities, shopping malls, etc. We have proposed an efficient solution for IFLS queries that prunes the search space significantly and thus speed up the query processing by reducing number of required total indoor distance calculations. We have evaluated the performance our proposed approach and compared it with the newly formed baseline, which shows that our approach achieves a speedup from 2.84× to 71.29× for synthetic data and 97.74× for real data over the baseline. In future, we plan to consider moving clients for IFLS queries.

9 ACKNOWLEDGEMENTS

Tanzima Hashem is supported by Bangladesh University of Engineering and Technology (BUET) Basic Research Grant. Muhammad Aamir Cheema is supported by Australian Research Council (ARC) DP230100081 and FT180100140. Hua Lu is supported by Independent Research Fund Denmark (IRFD) under no. 8022-00366B.

REFERENCES

[1] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. 1990. The R*-Tree: An Efficient and Robust Access Method for Points and Rectangles. In *SIGMOD*. 322–331.

[2] Zitong Chen, Yubao Liu, Raymond Chi-Wing Wong, Jiamin Xiong, Ganglin Mai, and Cheng Long. 2014. Efficient algorithms for optimal location queries in road networks. In *SIGMOD*. 123–134.

[3] Yu-Chi Chung, I-Fang Su, and Chiang Lee. 2018. k-most suitable locations selection. *GeoInformatica* 22, 4 (2018), 661–692.

[4] Jiangtao Cui, Meng Wang, Hui Li, and Yang Cai. 2018. Place Your Next Branch with MILE-RUN: Min-dist Location Selection over User Movement. *Inf. Sci.* 463-464 (2018), 1–20.

[5] Yang Du, Donghui Zhang, and Tian Xia. 2005. The Optimal-Location Query. In *SSTD*, Vol. 3633. 163–180.

[6] Zijin Feng, Tiantian Liu, Huan Li, Hua Lu, Lidan Shou, and Jianliang Xu. 2020. Indoor Top-k Keyword-aware Routing Query. In *ICDE*. 1213–1224.

[7] Yunjun Gao, Shuyao Qi, Lu Chen, Baihua Zheng, and Xinhan Li. 2015. On efficient k-optimal-location-selection query processing in metric spaces. *Inf. Sci.* 298 (2015), 98–117.

[8] Yunjun Gao, Baihua Zheng, Gencai Chen, and Qing Li. 2009. Optimal-Location-Selection Query Processing in Spatial Databases. *TKDE* 21, 8 (2009), 1162–1177.

[9] Jin Huang, Zeyi Wen, Jianzhong Qi, Rui Zhang, Jian Chen, and Zhen He. 2011. Top-k most influential locations selection. In *CIKM*. 2377–2380.

[10] Tiantian Liu, Zijin Feng, Huan Li, Hua Lu, Muhammad Aamir Cheema, Hong Cheng, and Jianliang Xu. 2023. Towards Indoor Temporal-Variation Aware Shortest Path Query. *IEEE Trans. Knowl. Data Eng.* 35, 1 (2023), 998–1012. https://doi.org/10.1109/TKDE.2021.3076144

[11] Tiantian Liu, Huan Li, Hua Lu, Muhammad Aamir Cheema, and Lidan Shou. 2021. Towards Crowd-aware Indoor Path Planning. *PVLDB* 14, 8 (2021), 1365–1377.

[12] Yubao Liu, Zitong Chen, Ada Wai-Chee Fu, Raymond Chi-Wing Wong, and Genan Dai. 2021. Optimal location query based on k nearest neighbours. *Frontiers Comput. Sci.* 15, 2 (2021), 152606.

[13] Hua Lu, Xin Cao, and Christian S. Jensen. 2012. A Foundation for Efficient Indoor Distance-Aware Query Processing. In *ICDE*. 438–449.

[14] Jianzhong Qi, Rui Zhang, Lars Kulik, Dan Lin, and Yuan Xue. 2012. The Min-dist Location Selection Query. In *ICDE*. 366–377.

[15] Chaluka Salgado. 2018. Keyword-aware Skyline Routes Search in Indoor Venues. In *ISA*. 25–31.

[16] Chaluka Salgado, Muhammad Aamir Cheema, and Tanzima Hashem. 2019. Continuous Detour Queries in Indoor Venues. In *SSTD*. 150–159.

[17] Chaluka Salgado, Muhammad Aamir Cheema, and David Taniar. 2018. An efficient approximation algorithm for multi-criteria indoor route planning queries. In *SIGSPATIAL*. 448–451.

[18] Zhou Shao, Muhammad Aamir Cheema, and David Taniar. 2018. Trip Planning Queries in Indoor Venues. *Comput. J.* 61, 3 (2018), 409–426.

[19] Zhou Shao, Muhammad Aamir Cheema, David Taniar, and Hua Lu. 2016. VIP-Tree: An Effective Index for Indoor Spatial Queries. *PVLDB* 10, 4 (2016), 325–336.

[20] Zhou Shao, Muhammad Aamir Cheema, David Taniar, Hua Lu, and Shiyu Yang. 2021. Efficiently Processing Spatial and Keyword Queries in Indoor Venues. *TKDE* 33, 9 (2021), 3229–3244.

[21] Tian Xia, Donghui Zhang, Evangelos Kanoulas, and Yang Du. 2005. On Computing Top-t Most Influential Spatial Sites. In *VLDB*. 946–957.

[22] Xiaokui Xiao, Bin Yao, and Feifei Li. 2011. Optimal location queries in road network databases. In *ICDE*. 804–815.

[23] Xike Xie, Hua Lu, and Torben Bach Pedersen. 2013. Efficient distance-aware query evaluation on indoor moving objects. In *ICDE*. 434–445.

[24] Lv Xu, Ganglin Mai, Zitong Chen, Yubao Liu, and Genan Dai. 2017. MinSum Based Optimal Location Query in Road Networks. In *DASFAA*. 441–457.

[25] Bin Yang, Hua Lu, and Christian S. Jensen. 2010. Probabilistic threshold k nearest neighbor queries over moving objects in symbolic indoor space. In *EDBT*, Vol. 426. 335–346.

[26] Donghui Zhang, Yang Du, Tian Xia, and Yufei Tao. 2006. Progressive Computation of the Min-Dist Optimal-Location Query. In *VLDB*. 643–654.