A connectivity index for moving objects in an indoor cellular space

ARTICLE *in* PERSONAL AND UBIQUITOUS COMPUTING · FEBRUARY 2013

Impact Factor: 1.52 · DOI: 10.1007/s00779-013-0645-3

4 AUTHORS, INCLUDING:

Sultan Alamri
Saudi Electricity Company
12 PUBLICATIONS **65** CITATIONS
SEE PROFILE

Maytham Safar
Kuwait University
116 PUBLICATIONS **821** CITATIONS
SEE PROFILE

Haidar Al-Khalidi
Monash University (Australia)
11 PUBLICATIONS **54** CITATIONS
SEE PROFILE

Noname manuscript No.
(will be inserted by the editor)

A Connectivity Index for Moving Objects in Indoor Cellular Space

Sultan Alamri · David Taniar · Maytham Safar · Haidar Al-Khalidi

Received: date / Accepted: date

Abstract With the current appropriate indoor positioning devices such as RFID, Bluetooth and WI-FI, locations of moving objects will be an important foundation for a variety of applications such as tracking moving objects, security and way-finding. Many studies have proven that most individuals spend their lives in indoor environments. Therefore, in this paper, we propose a new index structure for moving objects in cellular space. The index will be based on the connectivity (adjacency) between the indoor environment cells, which can effectively respond to the spatial indoor queries and enable efficient updates of the location of a moving object in indoor space. An empirical performance study suggests that the proposed indoor-tree in terms of measurements and performance is effective, efficient and robust.

Keywords index structure, indoor space, moving objects

1 Introduction

People spend the majority of their time in indoor environments, working, living, shopping and entertaining [21,14]. Hence, indoor environments have become increasingly large and complex. For instance, the Tokyo Metro has 274 stations and 13 lines and passengers numbering in excess of 8 million daily. Consequently, positioning and tracking of moving objects is an important research field with many applications including tracking moving objects, way finding, and security [27].

Current global navigation satellite systems such as GPS are not suitable for indoor environments (spaces). With the new positioning devices technologies for indoor spaces such as RFID, Bluetooth and Wi-Fi [7], large volumes of tracking data become available which provide a variety of services such as indoor navigation, objects tracking, security, and positioning. Indoor positioning technologies (e.g. Wi-Fi), have become more advanced [27,25]. Thus motivated, this paper provides a new indexing technique for moving objects in cellular indoor space.

In the past few years, much research has gone into the development of outdoor applications involving moving objects [21,14,26], and the indexing and querying of the trajectories of moving objects and their locations [28,3,23,20,5]. However, the outcomes of those researches are not suitable for indoor scenarios for the following reasons. First, indoor space is essentially different from outdoor space in many respects. The measurements in outdoor and indoor spaces are different. In outdoor space, Euclidean space or a spatial network is typically used; whereas, indoor space is related to the notion of cellular space.

Additionally, in indoor space, the environment contains different entities such as rooms, doors and hallways that both enable and constrain movement. As a result, the prevention or constraint of movement needs to be considered in the data structure for indoor spaces. Second, the positioning technologies differ for outdoor spaces and indoor spaces. In outdoor space, GPS is capable of continuously reporting the velocity and loca-

S. Alamri · D. Taniar · H. Al-Khalidi
Faculty of Information Technology, Monash University, Australia
E-mail: Sultan.alamri@monash.edu
E-mail: david.taniar@monash.edu
E-mail: Haidar.Al-Khalidi@monash.edu

Maytham Safar
Computer Engineering, Kuwait University, Kuwait
E-mail: maytham.safar@ku.edu.kw

tion of a moving object with varying accuracies. On the other hand, a proximity analysis is based on indoor positioning technologies which are not able to report the exact velocities or exact positions [10,14,33]. Examples of indoor positioning devices are RFID reader and Bluetooth which are based on the sensing or activation range of a positioning device.

In this paper, we propose a cells connectivity-based index structure for moving objects. Our index structure focuses on the moving objects based on the notion of cellular space, in contrast to the outdoor space structures which are based on the space domain, Euclidean or spatial network. Moreover, the key idea of our moving objects indexing is to take advantage of the entities such as doors and hallways that enable and constrain movement in an indoor environment. Therefore, we obtain an optimal representation of the indoor environment that is different from the outdoor environment. Moreover, if the indoor environment is seen in term of connectivity between the adjacent cells, then the spatial queries can be answered more efficiently.

In addition, the distance in our data structure is the number of hops of the cells instead of the Euclidean distance, that is used in some researches in indoor spaces [34,14]. Note that this work focuses only on the index structure with its constructions algorithms. We will leave the query processing for a forthcoming paper.
Our major contributions are:

- We develop a moving objects index structure for indoor space (indoor-tree) which can manage memory wisely via a neighbours' distance lookup table and significantly serve the traditional spatial queries in addition to queries related to the connectivity in indoor environments.
- We provide an indoor filling space algorithm to represent overlapping between the cells. The indoor space cannot precisely be transformed to a straight line (such as Hilbert space); therefore, we propose an expansion idea that provides an optimal representation of the filling indoor space.
- We present accompanying algorithms for the process of constructing the data structure for indoor space.

2 Related Work

The majority of the moving objects data structures are based on Euclidean distance and the availability of GPS-type positioning is either explicit or implicit. Data structures that focus on moving objects in outdoor spaces can be classified as follows:

Works that concentrate on *trajectories* of the moving objects [4,3,26]. We start with the Trajectory Bundle tree (TB-tree), which is based on the R-tree [24,11]. The idea is to index trajectories by allowing a leaf node to contain line segments only from the same trajectory, which assists in retrieving the trajectory of an individual object, but negatively influences the spatiotemporal range queries [4,21]. Another trajectory index is named STR tree (Spatio-Temporal R-tree) [4,3,26]. STR is based not only on spatial closeness, but also an trajectory preservation. STR was introduced in order to balance spatial locality with trajectory preservation. The main idea of the STR tree is to keep line segments within to the same trajectory [26]. SETI (Scalable and Efficient trajectory Index) [3] basically partitions space-dimensions into non-overlapping cells. The goal is to ensure that trajectory line segments in the same index partition belong to the same trajectory. Then, inside each partition, the line segments will be indexed by a separate spatial index (R-tree) [3,14].

Works that concentrate on *historical* moving objects [29,31,23]. Many applications such as road planning applications and security applications use the historical data of moving objects. The Historical R-tree (HR-tree) is one of the earliest data structures which concentrated on historical data [23]. The main idea is to use the timestamp history to construct the R-tree. R-trees can make use of common paths if objects do not change their positions, and new branches are created only for objects that have moved. It is clear that HR-trees are efficient in cases of timestamp queries, as search degenerates into a static query for which R-trees are very efficient. However, the massive duplication of objects can lead to large space consumption [29,23]. Another work that focuses on the historical data is the Multi-version 3D R-tree (MV3R-tree) [29] which basically uses Multi-version B-trees [24,21] and combines them with 3D R-trees [31]. The MV3R-tree includes large improvements which produce huge space savings without influencing the performance of the timestamp queries compared to the HR-tree. Another work focuses on the historical data but for a road network named Fixed Network R-tree (FNR-tree) proposed by Frentzos [8]. The idea is to take into account the road networks constraint. The FNR-tree contains a 2-dimensional (2D) R-tree and a 1-dimensional (1D) R-tree. The 2D R-tree is intended to index the spatial data of the network, whereas, the 1D R-tree is intended to index the time interval of each moving object.

Works that focus on the *future* and the current positions [28,20,5]. Saltenis et al. introduced the TPR-tree, (Time Parameterized R-tree) which is based on the R*-tree, in order to construct and manage moving objects.

The main idea of the TPR-tree is that the index stores the velocities of objects along with their positions in nodes. Moreover, the intermediate nodes' entries will store an MBR (minimum bounding rectangle), beside its velocity vector which they call VBR. As an extension of TPR-Tree, Yufei Tao proposed TPR*-tree [30] which develops the insertion/deletion algorithms in order to improve the performance of the TPR-tree. The TPR-Tree concept has many successors and several methods have been proposed to improve various aspects. Work such VTPR-Tree provide a good dynamic update performance and concurrency [19]. Moreover, some works such as [22] focus on indexing the moving objects in landmarks, where the tracking of moving objects in the topographical area is needed. This is different from the indoors, where the objects freely move inside the cells without any tracking.

Throughout the journey of the outdoor data structures, the availability of GPS-type positioning is explicit or implicit. Moreover, the majority of these works are based on Euclidean space or a spatial network is typically used. Whereas, indoor space is related to the notion of cellular space. The indoor space environment contains different entities such as rooms, doors and hallways that enable or constrain movement [10]. Therefore, outdoor data structures cannot be applied to indoor space, at least without reasonable enhancement. Indoor space needs to be fixed with indoor positioning devices, such as WI-FI, RFID reader and Bluetooth, which are based on sensing or activation. Furthermore, these positioning technologies determine the existence of moving objects at pre-defined cellular locations which is different in outdoor space. Besides, the limitation of the positioning device sensing ranges might cause large volatility of the data if outdoor data structures are used [33,7,14].

To the best of our knowledge, we are the first to build an index structure for the indoor environment that is to be based on connectivity between cells. Indoor environments are not based on Euclidean space; hence, treating the indoor environments based on their connectivity is the optimal way of building data structures for indoors.

3 Preliminary

3.1 Motivation

To illustrate the problem, we consider a database that records the position of a moving object in a coordinate base. For each object, we store an initial $[x, y]$, in the current time instant. Therefore, we can determine the moving objects location based on its coordinators on the floor. We assume that the objects are indexed by a metric data structure such as (TPR-tree). Moreover, the system is dynamic, i.e., objects may be deleted or new objects may be inserted. Table 1 explains the notations used throughout this paper.

Table 1 notations used throughout this paper

Notation	Definition
C	Cell
S	Space
P	Set of moving objects
tc	The current time
F	Floor number
$NX(Oi)$	Next cell of object Oi
$Dist(q, O)$	The Euclidean distance between query q and object O
$ActDist(q, O)$	The actual distance between query q and object O (with consideration to the constraints entities)
$MINDIST(q, O)$	Minimum distance between query q and O
$CellDist(Oi, Oj)$	The number of hops of the cells between Oi and Oj
$R(P)$	P indexed on metric structures such as R-tree.
$CU(P)$	P based on cellular base
$Ci \chi Cj$	Ci is connected to Cj
$Ci \dashrightarrow Cj$	Ci continue the movement from the DC toward the last cell.
$Ci(N\chi)$	The number of connection in Ci
MIN	Minimum values
RC	Range of expand points(cells)
LE	Largest expand point
DC	Default Cell
N	Leaf node
$EX-P$	*Expand point*
MBR	Minimum Boundary Rectangle
$[x, y]$	x and y coordinates of a point
v	velocity vector
$ChildPTR$	The pointer to the child node
PTR	The pointer
O_n	The maximum capacity of a leaf node

Let $P(tc) = [x1, y1]$ be the initial location of an object $O1$ at the current time tc. Then, the object changes its position which can be calculated as $P(tc) = [x(tc), y(tc)]$ $= [x1 + vx(t - tc), y1 + vy(t - tc)]$, note that $[vx, vy]$ is the velocity vector.

Now, we would like to answer a typical spatial query such as (kNN) of the form (based on Figure 1): "What is the 1NN objects to object $O1$"? Assuming that the query q perform at tc and the velocity vector is static at tc. Given a location of $O1 = [xi, yi]$ and its adjacent Objects $O2 = [xj, yj], O3 = [xr, yr]$ and $O4 = [xe, ye]$. Based on the metric properties that the objects structured on, the result will be $O2$. However, we can notice that $O2$ is not the right answer, because of the entities that enable and constrain the movement in an indoor environment (doors and hallways). Based on the actual distance ($ActDist(O1, O2) > Dist(O1, O2)$), $O2$ is the farthest one from $O1$, and $O3$ is the right result as 1stNN. The indoor environment is not an Euclidean space where objects can move freely without restrictions. Everything that restricts the movement must be considered to obtain the right results. Since the indoor space has these entries, and it needs to be fixed with multiple positioning devices, we argue that the connectivity between the cells is the optimal method of indexing the indoor space. Figure 2 illustrates the cell connectivity idea (where the number of hops between the cells is used instead of Euclidean distance, (see Figure 3)) giving more effective performance than do the metric structures. Moreover, indoor environments are not based on GPS, where the velocity and the location can be reported continuously. (Note that kNN queries illustrate the weakness of using a metric structure on indoor environments; *other spatial queries* can prove the same).

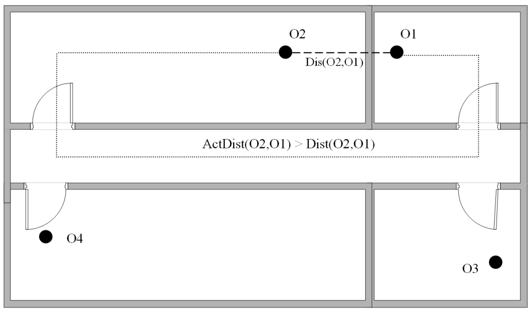


Fig. 1 Using $R(P)$ in indoor space can return wrong results, where in this example $O2$ is chosen as the 1stNN of $O1$, where the right result is $O3$ because of $ActDist(O1, O3) < ActDist(O1, O2)$.

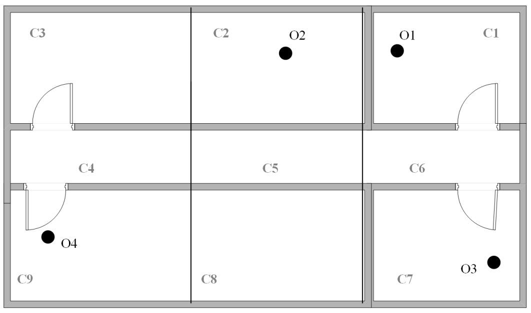


Fig. 2 An example of the cells' coverage and distribution which is more likely to return an accurate result based on the neighbours and connections between the cells

Definition 1: let $P = \{O1, O2, ..., On\}$ a set of moving objects in indoor space, If $R(P)$, the (1stNN) of $O1$ is $O2$ if $ActDist(O1, O2) < ActDist(O1, Oi))$ where $i \neq 2$.

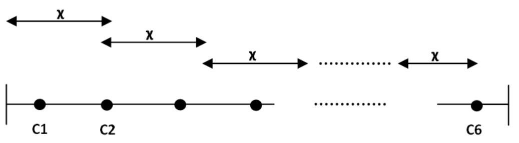


Fig. 3 The number of cells hops between $C1$ to $C6$ $CellDist(O1, O6)$

Definition 2: Given a set of moving objects
$P = \{O1, O2, ..., On\}$, If $CU(P)$, the $CellDist(Ox, Oy)$ is the number of hops of the cells, and calculated as:
$CellDist(Ox, Oy) = |\{C1, C2, ..., Cn\} - 1|$,
where Ox$\overset{in}{\rightarrow}$C1 *and* Oy$\overset{in}{\rightarrow}$Cn, $\forall$ C1χC2χC8.....χCn.

Definition 3: let $P = \{Oi, ..., On\}$ be a set of moving objects. In indoor cellular space, $O2$ is considered as the 1stNN of $O1$ if $CellDist(O1, O2) < CellDist(O1, Oj))$ where $j \neq 2$. :

Lemma 1: let $P = \{Oi, ..., On\}$ be a set of moving objects in indoor space, Oj has minimum distance to Oi $MINDIST(Oj, Oi)$, however, Ox is considered as the 1stNN of Oi iff:

– $ActDist(Oi, Oj) > Dist(Oi, Oj)$ and
– $ActDist(Oi, Ox) < ActDist(Oi, Oj)$.
– $CellDist(Oi, Ox) < CellDist(Oi, Oj)$,
$\forall$ j $\neq i$ and $j \neq x$.

Proof. Let $\{O1, O2, ..., On\}$ be the ordering of P moving objects at time tc, sorted by position coordinate (x, y). Assume we maintain a 1stNN query to $O1$. Based on definition 1, since $ActDist(O1, O2) > Dist(O1, O2)$ and $ActDist(O1, O3) < ActDist(O1, O2)$. Moreover, based on definition 2,3 where the $CellDist$ is the basis for the indoor spatial queries where $CellDist(O1, O3) < CellDist(O1, Oj))$ where $j \neq 2$, therefore; $O3$ is the 1stNN of $O1$.

3.2 Cellular Space

Indoor Space has restriction entities such as rooms, doors and walls; therefore, it is not possible to fix the whole indoor floor with one positioning device. Hence, the indoor floor will be fixed with many positioning devices and will be treated as cellular space. *The cellular space* is not based on any geometric representation of spatial property. Cellular space is a representation of the location by sets of cells that include spatial objects. The indoor space data structure has an important requirement related to the notion of cellular space.

Definition 4: Given a set of cells $C = \{C1, C2, ..., Cn\}$, space S and a set of spatial objects $\{O1, O2, ..., On\}$, the space is called *Cellular Space* iff:
$S = \bigcup Ci$, where Ci is the cell and
$\exists \;\; Cj$ such that $Oi \overset{in}{\rightarrow}$Cj.

Definition 5. Given a set of cells $C = \{C1, C2, ..., Cn\}$, and a spatial object Oi, Ci is *an adjacent cell* to Cj $(Ci\chi Cj)$ iff
Oi$\overset{in}{\rightarrow}$Cj *and* NX(Oi) *is* Ci.

The basic difference between cellular space and Euclidean space is the dependence of geometric representation of spatial property [33,35,14]. A query in outdoor space is given with coordinates such as (xi, yi) and (xj, yj). On the other hand, the queries in indoor space are usually based on cellular notations such as "What are moving objects in room 431?". In this example, the room number represents a cell identifier, which is the main difference from the outdoor coordinates in Euclidean space [17,10,12].

To illustrate further, take as an example a location within a train which is an indoor space. The location is identified by the wagon and seat numbers and not by its coordinates [17,13,25]. For more explanation, in a geometric space both, the located objects and locations will be represented as coordinate n-tuples, represented as points, areas, and volumes [21,32,18]. Basically, this type is based on reference coordinate systems (RCS) such as TB-tree, 3D R-tree and so on. On the other hand, the location of a moving object is indicated by abstract symbols or cells in cellular space. In cellular space, the location descriptions are represented by sets, and a located object in that case is considered as a member of these sets. Next, we illustrate how a moving object is represented in the cellular space.

Definition 6. The moving object O in cellular indoor space is represented as $\{O, (C, t)\, | C \in S, t = [Time], F\}$, where O is the moving object, t indicates the current time, C is the cell, S is space and F is the floor number.

The space will be divided into grids based on its positioning devices' coverage (Wi-Fi or RFID) (assume that the coverage will be grid). The division of the cells will depend on the technical resources of the positioning devices' coverage and on the partitions (walls and obstacles). Figure 4 represents an example of the coverage cells distribution.

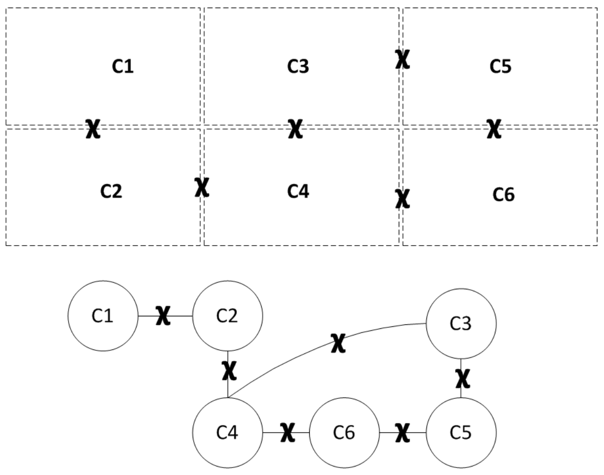


Fig. 4 An example of the coverage cells distribution, where χ means connected

4 Indexing Moving Objects in Cellular Space

In this section, we start by explaining the representation of cellular indoor space. Then we define the uniqueness of our structure which is the *connections* (adjacency) in the indoor space cells with its structure and algorithm. Then, the *Indoor Filling Space Representation algorithm*, which is the structure that is based on the connections, facilitates the tree construction and the insert and delete algorithms. Subsequently, we represent the construction of the tree and the maintaining algorithms.

For illustration, we present the next example of a floor plane of 7 rooms including stairs $R7$ and corridor $R6$. Figure 5 illustrates the floor space. Each venue will be divided based on the sensors' coverage. For example, Room 6 which is the corridor will be divided into 5 cells ($C14$, $C10$, $C5$, $C2$ and $C1$) as shown in Figure 6. Note the stairs will be treated as a room ($R7$ is the stairs). The result of the new cells distribution is as shown in Figure 6.

The indoor space is an overlapped environment which has many constraints such as rooms, doors and hallways. We assume that the overlap in the positioning devices' coverages is addressed by creating a new individual cell for any cells overlap. Since the overlapped cells issue in indoor spaces is a considerable research

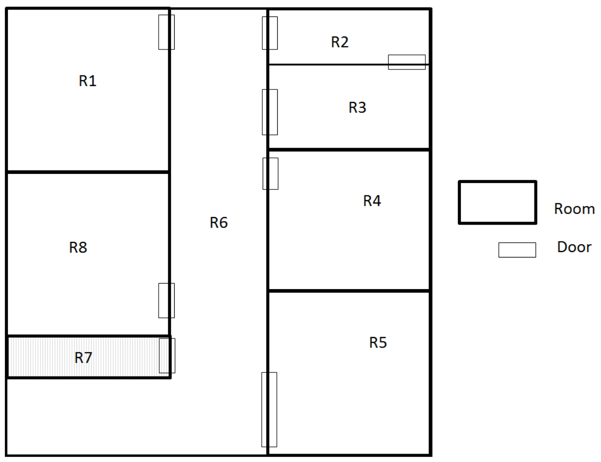


Fig. 5 An example of floor plane of 7 rooms

area, we will leave this detail for the future work.

From the cells' distributions, it is clear that for any object located in a particular cell, its next location must be in one of its adjacent cells. For example, (Figure 6) for objects located in $C12$, their next location must be $C16$. Therefore, we can establish connections between the cells which indicates the possible movements between the cells.

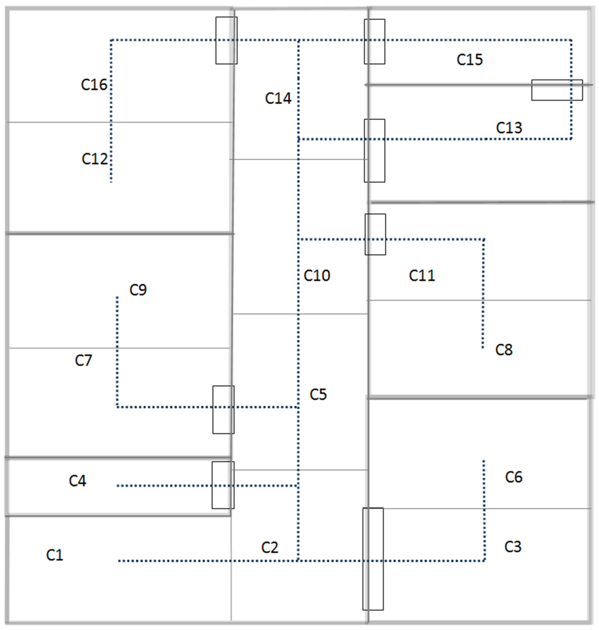
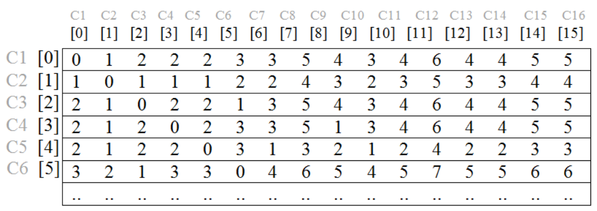


Fig. 6 Illustration of the connectivity between the cells

The connection between the cells will be stored in a neighbours' distance lookup table to facilitate the grouping in the data structure. In this table we have pre-computed neighbours' distances between cells. The distance is not metric, but based on the number of hops, where 0 means the cell itself, 1 means neighbour number one, 2 means neighbour number two and so on. We assume that the positioning devices' coverage and the neighbours distance lookup are pre-fixed. The neighbours' distance lookup table is established to assist in building *the Indoor Filling Space Representation algorithm.* Moreover, it will be used to compare the adjacency of the cells to obtain the adjacent cells in case of inserting, deleting or updating (to determine the suitable nodes). Note that the size of the neighbours' distance lookup table is small (Figure 7). Moreover, we store the cells connection as a multi-dimensional array list where the search complexity is $O(n)$[1]. The checking algorithm that will use the table is called the Adjacency Comparison algorithm, which is used to return the nearest connected cell (the cell that has the MIN value in neighbours' distance lookup table).



		C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14	C15	C16
		[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]	[15]
C1	[0]	0	1	2	2	2	3	3	5	4	3	4	6	4	4	5	5
C2	[1]	1	0	1	1	1	2	2	4	3	2	3	5	3	3	4	4
C3	[2]	2	1	0	2	2	1	3	5	4	3	4	6	4	4	5	5
C4	[3]	2	1	2	0	2	3	3	5	1	3	4	6	4	4	5	5
C5	[4]	2	1	2	2	0	3	1	3	2	1	2	4	2	2	3	3
C6	[5]	3	2	1	3	3	0	4	6	5	4	5	7	5	5	6	6
		..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..

Fig. 7 The neighbours' distances

Algorithm 1 Adjacency Comparison algorithm

```
1: /* check the inserted to $Ci$ with set of $C$. */
2: /* Adjacency Comparison Algorithm will return the cell
   that has the $MIN$ value */
3: R = initial value
4: for $Ci$ adjacent cells AC.i do
5:     if AC.i < R then
6:         R = AC.i
7:     end if
8: end for
9: Return $R$ // MIN value
```

4.1 Indoor Filling Space Representation

Indoor space is a filling space based on connection between the cells [10,17]. Since indoor space is usually based on cellular notation, unlike outdoor space which is based on coordinates (xi, yi), we need to establish a pre-computed indoor filling space algorithm (extracted from the neighbours' distance table) to represent overlapping between the cells. Since, the indoor space cannot be precisely transferred to a straight line, the expansion idea can assist us to represent the filling indoor space cells to determine the higher cell (has more connection) and the lower cell (has less connection). Note that the default cell DC is the cell that is chosen to be the main cell in the indoor floor.

Definition 7. Given a set of cells $C = \{C1, C2, ..., Cn\}$, Cj is considered as an expand point *EX-P* iff Cj ⇢Cn *as* C1⇢C2,C2⇢Cj, ...,Cj⇢Cn *where* C1 *is* DC *and* Cn *is the last cell*, ∀ C1χC2....CjχCn.

Definition 8. Given a set of cells $C = \{C1, C2, ..., Cn\}$, the expand points' order are $C1 > C2,...,> Cj$, iff:
- $C1$ is the DC and Cn is the last cell, and
- $C1$ ⇢C2,C2⇢Cx, ...,Cj⇢Cn, ∀ j $\neq$ x *and* x $\neq$ n.

Since the cells in the indoor space overlap, as shown in Figure 4 the cells connection is represented as an acyclic graph, from the default cell DC, we convert (based on definition 7,8) the acyclic graph to tree [15, 9]. Figure 8 shows as example of a tree acyclic graph converted to tree. The indoor expansion will be explained next.

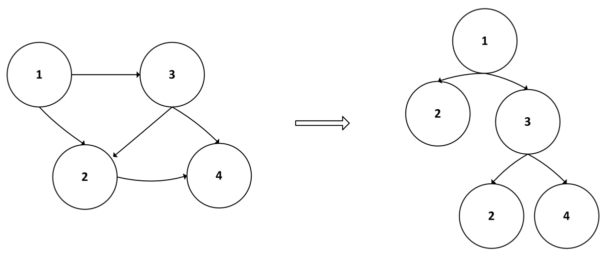


Fig. 8 Converting acyclic graph to tree

Our objective is to use the data of the expansion idea (stored similarly to the neighbours' distance table), in the construction of the tree and the maintaining operations such as insertion and deletion. It is clear that each cell must be connected to an expand cell (point); therefore, we will take advantage of this and record the two expand points with non-leaf nodes as range and the largest expand point will be recorded in the leaf nodes (for comparison and to choose node goals). The indoor filling space (expansion) algorithm steps are as follows:

- The algorithm will establish *expand points* which are the points (cells) based on definition 7, $C1$ ⇢ $C2$, $C2$ ⇢ Cj,...,Cj ⇢ Cn.
- The expansion will start from the default cell DC (assigned as an expand point) to the adjacent cells, where the algorithm will start to list the cells that have fewer adjacent cells. E.g. based on Figure 9, $C4(N\chi) < C3(N\chi)$.
- Then the cells that have more adjacent cells (more connections). E.g. based on Figure 9, $C5(N\chi) > C3(N\chi)$.
- The process is repeated up to the *last cell* (each cell that continues the connection will be considered as an *EX-P*).

Referring to Figure 9, assuming that $C1$ is the default cell, the algorithm will start from $C1$ and assign it as an *EX-P*. Then this will expand to the adjacent cells starting from the cells that have fewer connections ($C4$) to the cells that have more connections ($C3$ and $C5$). Cells that continue the connection will be assigned as expand points (e.g. $C3$ and $C5$). The highest expand points start from the left in descending order to the right. The ordering is logical because the moving objects start the movement from the default cell (the highest ($C1$)) to the last cell in the floor (in our case $C12$). Note that this expansion does not restrict the movement of the objects; they can still freely move between any cells. The objective of this is to have better understand the cells' overlap and connection. Then, we will take advantage of expand points and record the two expand points (RC) with non-leaf nodes as range and the largest expand point (LE) will be recorded in the leaf nodes (for comparison and to choose nodes' goals). Figure 9 represents the expand steps of the floor example.

As mentioned, an indoor environment is characterized by entities that enable or constrain the movement (doors and hallways). Hence, we take advantage of this in the indoor environment to build a data structure based on the connectivity between the cells. Traditional data structures such as R-trees and their followers based their comparison on Euclidean space, so the objects will be grouped together based on MBR least enlargement [11]. Moreover, other traditional data structures such as Hilbert R-trees and their followers based their comparison on the Hilbert value, so the objects are grouped together based on MBR based on LHV (Largest Hilbert Value) [16]. These techniques cannot be applied to an indoor environment which is not based on Euclidean distance. Furthermore, the indoor environment has overlapping areas, which cannot be treated as straight lines as are the Hilbert R-trees. Therefore, the expansion idea provides an optimal representation of the indoor space, which helps us to compare cells and group the objects based on their connections.

4.2 Tree Construction

Based on the adjacency concept mentioned in the previous section, the data structure will group the entries based on their adjacency cells. This is the best means

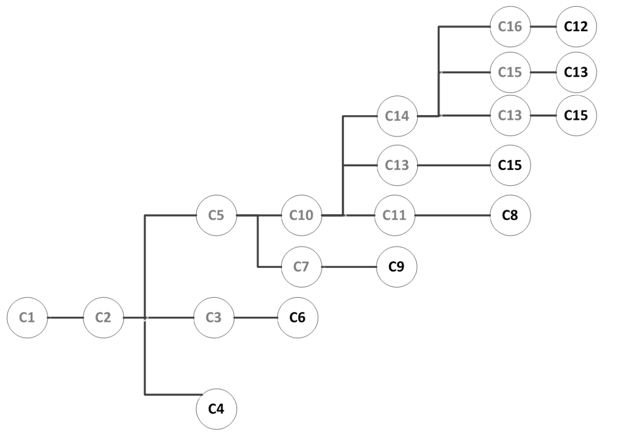


Fig. 9 The expand steps of the floor example (grey indicates the expand points)

of measuring the indoor environment which is based on the adjacency and connections between venue and rooms. Conversely, the outdoor environment is based on metric measurement. Therefore, the idea is to start by grouping the objects inside the same cell. In the case of overflowing MBR, splitting will be performed to group it with one of the objects in adjacent cells based on the adjacency comparison algorithm. The idea is to check the RC (Range cells) at the non-leaf node or the LE (the largest expand point) at the leaf node by comparing them with the inserted cells (by the adjacency comparison algorithm) and choosing the cell that has the MIN value (the nearest connected cell).

Definition 9. Given a set of cells $C = \{C1, C2, ..., Cn\}$, and set of non-leaf nodes $N = \{N1, N2, ..., Nn\}$, the RC *(Range cells)* in Ni is the two expand cells as follows:

– ⌊Ci⌋ is highest expand point and ⌈Cj⌉ is lowest expand point, where $Ci > Cx$,...,$> Cj$, ∀ Ci, Cx,...,Cj ∈ Ni.

Definition 10. Given a set of cells $C = \{C1, C2, ..., Cn\}$, and set of leaf nodes $N = \{N1, N2, ..., Nn\}$, the LE is highest expand point ⌊Ci⌋, where $Ci > Cx$,...,$> Cj$, ∀ Ci, Cx,...,Cj ∈ Ni.

Example: For the non-leaf nodes, using the data in Figure 11, the non-leaf node $N1$ has three leaf nodes $N1a$, $N1b$ and $N1c$, and six cells = {$C12$, $C16$, $C14$, $C13$, $C10$ and $C8$}. The RC for $N1$ is ($C10$; $C16$)(where $C10$ is the highest expand cell contained in $N1$ and $C16$ is the lowest). For the leaf nodes, using the data in Figure 12, the leaf node $N1a$ has two cells = {$C12$, $C16$}. The LE for $N1a$ is ($C10$)(where $C10$ is the highest expand cell contained in $N1a$).

In our indoor tree, the data then is ordered according to the connectivity between the cells. There are two main properties of an indoor-tree:

1. Non-leaf nodes contain (RC and $ChildPTR$) where RC as defined in definition 9. We store the RC in each non-leaf node, which is based on the indoor filling space algorithm (grey in Figure 9). Thus, each non-leaf node will have a range of two expand points as the maximum and minimum cells. $ChildPTR$ is the pointer to the child node. A non-leaf node contains at most O_n entries, which is the maximum capacity of the non-leaf nodes.

2. Leaf nodes contain (LE, obj, and PTR) where LE as defined in definition 10. Therefore, we record one expand cell only in the leaf node which is the largest connected cell at the node. For example, assume that a leaf node has three cells Ci, Cj and Cx where $Ci > Cj > Cx$ (three cells that contain the objects at that leaf node), we record Ci at the LE, because Ci is the largest expand point at that leaf node. The objects that are contained in the MBR at that time are denoted as obj, PTR is the pointer. A leaf node contains at most O_n entries, which is the maximum capacity of the leaf. The structure is illustrated in Figure 10.

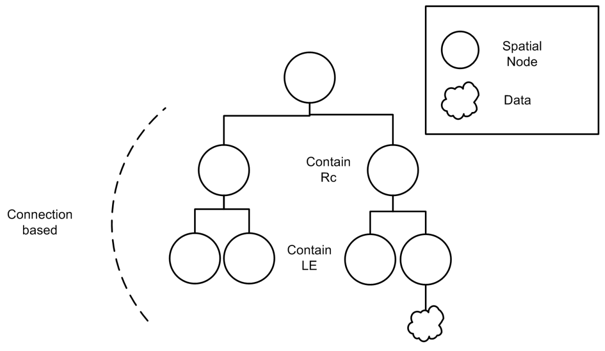


Fig. 10 Indoor-tree

Using the sample data in Figure 11, suppose that the five MBRs are clustered into larger MBR, where moving objects = $1, 2, 3, 4, ..., 12$ and $M = 3$ and $m = 2$. Note that the objects' positions are shown only for simplification purposes. Furthermore, $N1$ RC is ($C10, C16$)(where $C10$ is the highest expand cell contained in $N1$ and $C16$ is the lowest) and $N2$ RC is ($C1, C3$) based on the indoor filling space (expansion) algorithm. In the leaf node $N1a$ LE is ($C16$), $N1b$ LE is ($C14$), $N1c$ LE is ($C10$), and $N2a$ LE is ($C1$), $N2b$ LE is ($C3$). The indoor-tree is shown in Figure 12.

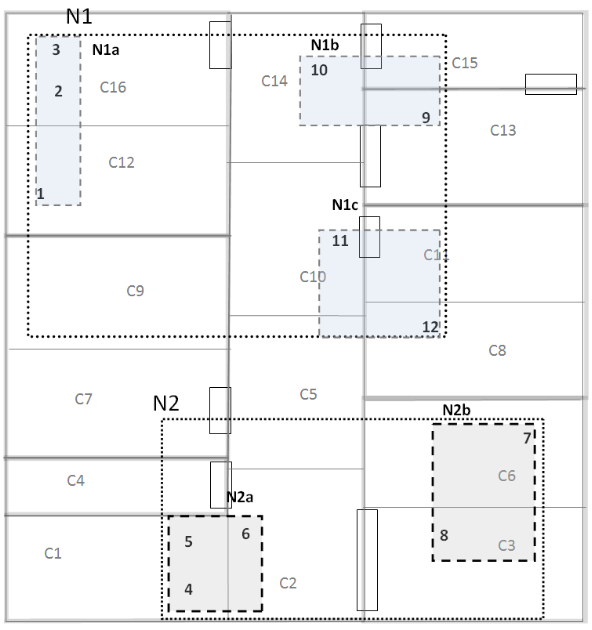


Fig. 11 The MBRs grouping based on the connection

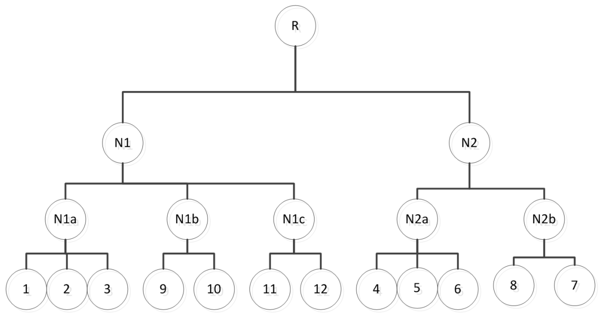


Fig. 12 An example of an indoor-tree

4.3 Insertion, Deletion and Updating

In our data structure, the splitting policy that will be used is called "immediate split policy". When an entry is inserted into a cell and the node overflows, the node has to be split immediately but to be grouped to the adjacent cells.

Definition 11. Given a set of Nodes $N = \{N1, ..., Nn\}$, and spatial objects $O1$,$O2$,...,On, where Ni contain at most m entries. Ni is indicated as an *overflow* node if Ni contain $> m$ entries.

Definition 12. Given a set of Nodes $N = \{N1, ..., Nn\}$, and spatial objects $O1, O2$,...,On, where Ni contains at most m entries. Ni is indicated as an *underflow* if Ni contain $< m/2$ entries.

The *choose leaf node algorithm* starts to check the range of the cells (RC) (for the non-leaf nodes) and compares it with the inserted cell. If the inserted cell is one of the range cells, we choose the node that has this RC; otherwise, we check the inserted cell with the each range (RC) (called Adjacency Comparison Algorithm) and choose the one that has a MIN value cell. For the leaf node, the algorithm will check the inserted cell with the LE and choose the node that has the MIN value cell. For example, using the sample data in Figure 11, assuming that an entry '13' will be inserted to $C16$. The algorithm will start with the *choose leaf algorithm*, where we need to identify which non-leaf node is suitable for the new entity. Then *choose leaf algorithm* will call the adjacency comparison algorithm to compare $C9$ with the RC for each non-leaf node, hence it chooses $N1$. However, the non-leaf node $N1$ is overflowing which will lead to an immediate split. $N1$ will be split to $N1^{'}$ and $N1^{''}$. Moreover, the algorithm will choose the leaf node $N1a$ (result of comparing LE with $C16$) which is overflowing as well. $N1a$ will be $N1a^{'}$ and $N1a^{''}$. The non-leaf node splitting and the leaf node splitting are shown in Figure 13. If the RC comparison results and LE comparison results are equal, the algorithm will choose the node that has more neighbours (connections)(algorithm 2).

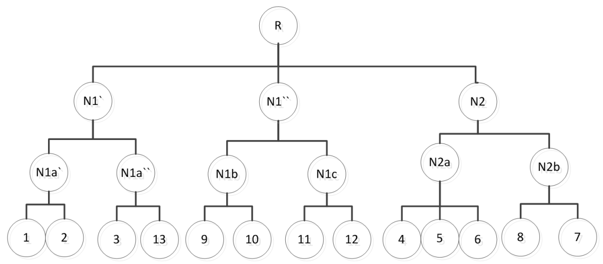


Fig. 13 Inserting 13 to $C16$

In the deletion operation, we check for the underflown node, instead of the overflown node. If a node is underflown, we need to check the sibling nodes that are based on the connection to participate in solving the underflown. The deletion algorithm will be explained in the next example. Suppose that we want to delete object 7 from $C6$ where $m = 2$ and $M = 3$ (Figure 12). The algorithm will first call the FindLeaf algorithm which iteratively searches for the target node from the root node to the leaf node [30,11]. After discovering that $N2b$ is underflown, the delete algorithm will determine the sibling-connected node based on adjacency comparison algorithm (Figure 14). Then re-insert the remain entities into that node (if it is overflown we split, as explained in the insertion algorithm). Hence, in the

Algorithm 2 ChooseLeaf algorithm

```
1: /* Return the leaf node that has the inserted $n$ */
2: Set $N$ to the root node
3: if $N$ is non-leaf node then
4:     Call adjacency comparison algorithm(ACA) compare with RC
5:     Node($N_i$) has $MIN$ Value
6:     Choose the ($N_i$,ptr)
7:     if $N_i$.value = $N_j$.value then
8:         Call adjacency comparison algorithm(ACA)
9:         Check $RC$ at $N_i$ AND $RC$ at $N_j$
10:        if $RC.N_i$ adjacents > $RC.N_j$ adjacents then
11:            Choose the ($N_i$)
12:        else
13:            Choose the ($N_j$)
14:        end if
15:    end if
16: end if
17: if $N$ is leaf node then
18:    Call $ACA$ compare with LE
19:    Node($N_x$) has MIN Value
20:    Choose the ($Nx, ptr$)
21:    if $N_x$.value = $N_y$.value then
22:        Call adjacency comparison algorithm(APA)
23:        Check $LE$ at $N_x$ AND $LE$ at $N_y$
24:        if $LE.N_x$ adjacents > $LE.N_y$ adjacents then
25:            Choose the ($N_x$)
26:        else
27:            Choose the ($N_y$)
28:        end if
29:        if $LE.N_x = LE.N_y$ then // (overlapping case )
30:            Choose the Node that has fewer entities
31:        end if
32:    end if
33: end if
```

Algorithm 3 Insert algorithm

```
1: /* Input: $o$ is the entry to be inserted. */
2: procedure INSERT($o$ into $C$)
3:     Insert $o$ into the $C$
4:     call ChooseLeaf to find the suitable leaf node $N$
5:     if $N$ overflows then
6:         Choose the new $N'$ based on $ACA$
7:         Split $N$
8:         Group $o$ in $N'$
9:         Update $RC$,$LE$ at $N$,$N'$.
10:    end if
11:    if node split propagated to the root node and root node to split then
12:        Create a new root node with new resulting child nodes
13:    end if
14: end procedure
```

deletion, if underflow occurs, we deal with the adjacent cells.

Although the nodes have some sort of adjacency (connection) in an indoor-tree, in the search process, the adjacency is not used. Therefore, the search process in indoor-tree follows the same search process in the original R-tree [11]. So, basically the search starts from the root node, and descends the tree examining all nodes that intersect the query scope. As made clear, one essential advantage of our data structure is that the grouping of the objects is based on the connectivity between the cells. Therefore, it is more likely that objects will move from a cell to another without any updating in the nodes (since the node contains the adjacent cells).

Algorithm 4 delete algorithm

```
1: /* Input: $o$ is the entry to be deleted. */
2: procedure DELETE($o$ from $C$)
3:     delete $o$ from the $C$
4:     FindLeaf to find the leaf node $N$
5:
6:     if $N$ underflows then
7:         find $N$ sibling adjacent based on $ACA$
8:         re-insert $o$ into $N_i$
9:         if $N_i$ overflows then
10:            HandleOverflow($N_i, o$)
11:        end if
12:    end if
13:    Update $RC$,$LE$ at $N$,$N_i$.
14:    if node split propagated to the root node and root node to split then
15:        Create a new root node with new resulting child nodes
16:    end if
17: end procedure
```

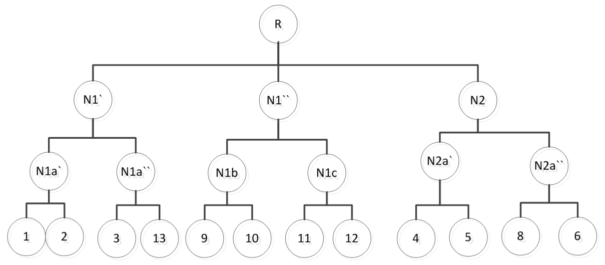


Fig. 14 Deleting 7 from $C6$

5 Experimental Results and Performance Analysis

In this section, we present our experimental results to evaluate the proposed index via the query performance of the indoor-tree index structure. The experiment has been carried out on an Intel Core i5-2400S processor 2.50GHz PC, with 4 GB of RAM running on 64-bit Windows 7 Professional. The maximum number of entries per node, M, was 60 and the minimum m 30. The data structure has been implemented in Java. The data set size will start from 10 to 1000 moving objects on the indoor floor.

In this experiment, due to the lack of real data for indoor environments, we use synthetic datasets of moving objects on a floor of indoor space. We generate the

Algorithm 5 update algorithm

```
/* Input: $o$ is the entry to be updated. */
procedure UPDATE($o$ from $C$)
    FindLeaf to find the leaf node $N$
// the FindLeaf function
    if $o$ move to adjacent cells then
        positioning device skip updating
    end if
    if $N$ effected then
        if $N$ underflows then
            ChooseLeaf function to find $N$ sibling adjacent
            re-insert $o$ into $N_i$
        end if
        if $N_i$ overflows then
            HandleOverflow($N_i, o$)
        end if
    end if
    if node split propagated to the root node and root node to split then
        Create a new root node with new resulting child nodes
    end if
end procedure
```

location of the objects based on the number of cells. For example, in the 12-cells case, we generate the moving objects to be covered by all the cells, then we start to increase number of the objects in each cell. As mentioned earlier, the movement of the objects will be unspecific inside the cells (treated as static) and we update only when the object moves out of the cell and checks into a new cell. For the indoor environments, we use several indoor spaces, starting from 12-cells real case of floor environment, then extending the floor to 50, 100 and 150 cells. The cells expansion is based on realistic scenarios. It is clear that each case will have different expand cells, where the 12-cells case will have fewer expand cells than the others.

We investigate the following: tree construction costs and search and insert performance. For the former, the execution time is measured for each test. For the tree construction test, we will measure the costs for the different intensities of moving objects (number of the moving object) and the indoor space overlapping intensity/the connection complexity. For the query performance and maintaining operations test, we will measure the complexity and the efficiency for different intensities of moving objects, the influence of the indoor connection complexity and the expand points complexity. Note that the operations are performed 5-7 and the average is calculated. The parameters used are summarized in Table 2.

Table 2 Parameters and Their Settings

Parameter	Setting
Node capacity	60
Number of moving objects	(10 to 1000)
Indoor space	12,50,100,150 cells
Expand cells	increase with the increasing of the indoor cells
Operations	Insert, find
Dataset	synthetic

5.1 Tree Construction

Tree construction costs are illustrated in Figures 15,16. We generate moving objects starting from 10 until 1000 on each floor, then, we run the structure tree to group the moving objects together based on the indoor connectivity index. Figure 15 illustrates the increase of the moving objects for each indoor cells case. As the number of moving objects increases, the construction cost increases accordingly. This behaviour is natural in an indexing tree where, with the increase of the moving objects, the construction cost will increase, and the number of the nodes and splits will increase; moreover, the comparison and checking of the lookup table for the MIN values will be increased. Figure 16 illustrates the increase of the number of the cells. We can notice that the tree construction cost increases slightly in some of the cases, which indicates that the indoor tree index strategy is effective for the indoor tree construction.

Figure 17, indicates the effect of the cells connections' complexity. Basically, we tested the construction on different complexity of connections.Apart from the actual connection in the floor, we include the worst case connection which is a complete graph connection, and the best case is ascending connections [9]. Note that the increase of the complexity of the cells' connections does not impact on the tree construction costs. We can explain this behaviour as follows: when we insert any objects into any cell, with the increase of the connections, it will be more likely to have an adjacent cell to be grouped with, which does not usually occur in the ordinary case. Consequently, the proposed index tree construction cost remains steady and stable.

5.2 Search and Insert Costs

Next, we study the search and the insert costs. For the search, we test the typical descending search which is basically a search from the root to the leaf nodes searching for a particular object. For the insert we measure the proposed insert algorithm with its associated operations cost. Moreover, we consider the effects of in-

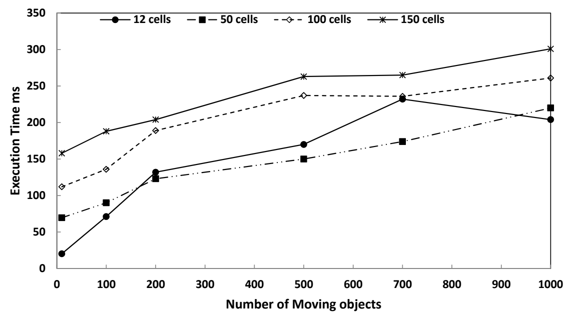


Fig. 15 Effect of Objects Number

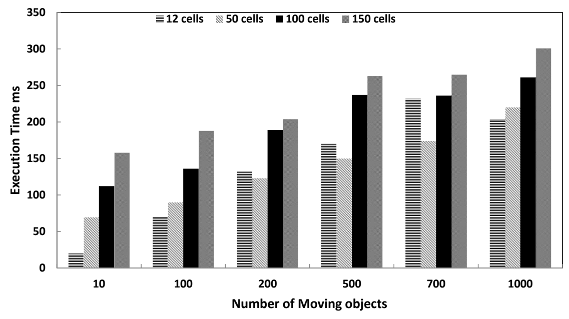


Fig. 16 Effect of Cells Numbers

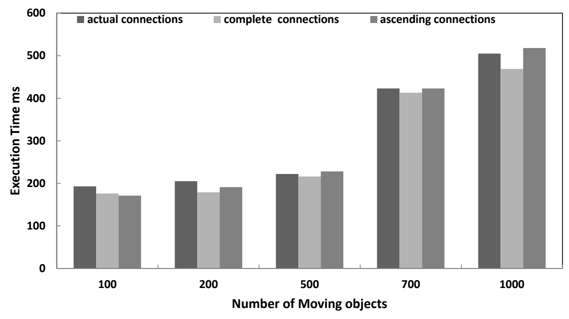


Fig. 17 Illustrates the effect of connection complexity

creasing the number of moving objects and the complexities of the connections and expand points. In this section, the test of the search and the insert is done after constructing the moving objects, where we search for particular objects in object densities of 10, 100, 200, 500, 700 and 1000. The insertion case is similar. In order to be able to report meaningful outcomes, we optimized the system's configuration and warmed up the system's caches by the execution of queries 5-7 times and the stabilized runtime will be used.

Effect of Objects Number. We use a different number of cells in order to monitor the influence of the increase in the number of moving objects for different cases. First, we measure the search query, then the insertion. For the search (find) query, we can notice in Figure 18, that with the increase of the moving object, the query cost increases slightly around 1.4ms. Moreover, we can notice that for the 12-cells case, when we perform query at 1000 density, the cost is almost similar to that for other densities. Note that the search query cost is usually less than the insertion, due to the related operations that can be produced from the insertion (such as split,compare,...,etc).

For the insertion, we insert an object into a particular cell several times, using objects of densities 10 - 1000 objects, and then we record the average time for each moving objects' density. We can notice in Figure 19 that in some cases, with the increase of the moving objects, the cost increases around 8ms. From Figure 19, we can notice that the insertion cost 12-cells case, increases slightly with the increase of the moving objects, due to the increase of the the related operations and nodes. However, we can also notice that for a density of 500 objects in 12 cells, the performance is close to that of the 500 objects density of 50 cells case.

Furthermore, Figure 20 illustrates the updating of the indoor-tree of 50 cells at density 1000 objects. We update 100 objects first, then 500 objects, then all moving objects. We can notice that our indoor-tree still performs efficiently when updating large densities of moving objects.

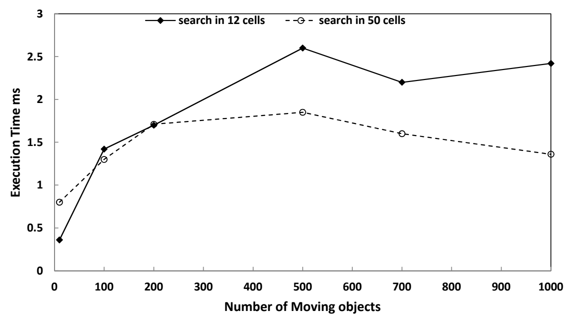


Fig. 18 Effect of Objects densities (Search Performance)

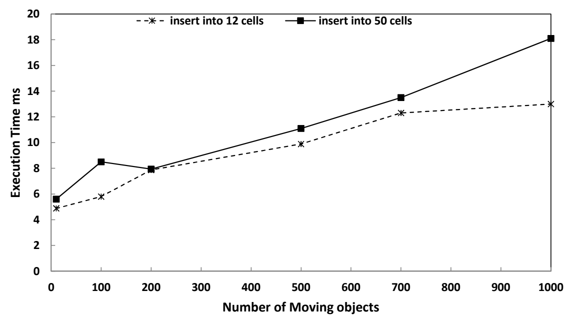


Fig. 19 Effect of Objects densities (Insert Performance)

Effect of Connections Number. Here, we measure the complexity based on the cells connections, which

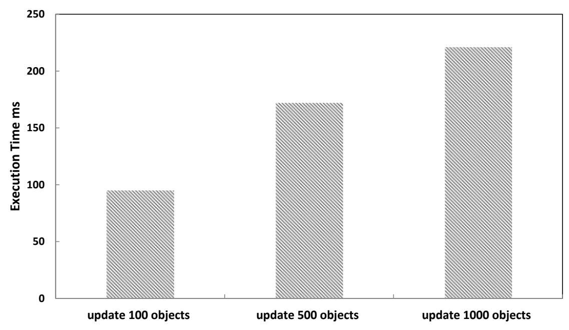


Fig. 20 Illustrates the effect of different updated Objects densities

means we increase the cells' neighbours in order to monitor the effect. Moreover, it illustrates the effect of the distributions of the cells in different densities. In this section, after examining the original connection case (we called it *actual connection*), we tested the worst case and the best case theoretically of the cells connection. The theoretical worst connection case, is a complete graph, where each cell is connected with all the other cells. For the best theoretical case, is the ascending connection, where each cell is connected with one other cell.

For the find query, the number of connections will not affect the search (find) query, because the indoor-tree is already constructed and the connection number will not be used in the search queries. Therefore, we tested only the insert, where the connection is essentially important. We start by inserting an object into a particular cell several times, and then we record the average time for each moving objects density (this will be done in all connections cases). We can notice from Figure 21 that with the increase in the number of moving objects, the insert cost in the worst case and the best case scenario is less than the actual connection. We explain this as follows: When a floor has more connections between its cells, this facilitates the insertion in our indoor-tree. The reason is that when an object is inserted into the floor, the choose leaf algorithm will check the inserted cell and compare it with the *RC* and *LE* of other nodes, where it is more likely to have many options, where we group it with any of its many neighbours. Moreover, the splitting can also be facilitated in environments with more connections , where any node has to be splitted (because of overflow or underflow), the objects will be grouped with the connected cells' entries. Figure 22 illustrates the updating of a high density of moving objects where we can notice that the indoor-tree still performs well in the worst case connection.

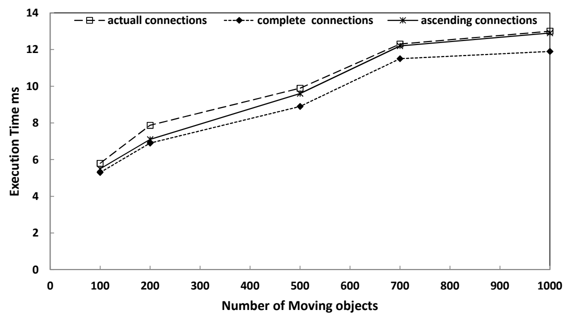


Fig. 21 The effect of connection complexity on inserting

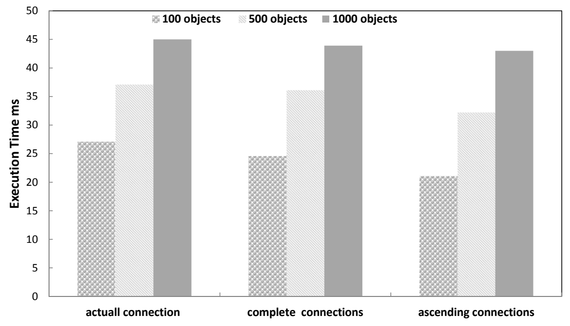


Fig. 22 The effect of connection complexity on updating Objects on different densities

Effect of Expand Points Number. Here, we measure the expand points' complexity. In this section, after examining the original expand points case (we call it *actual expansion*), we tested the *high expansion* where all of the cells are considered as expand points.

We tested the effect of the expansion complexity for the 50-cells densities. We can notice from Figure 23 that the updating of high density of moving objects, the indoor-tree still performs very well in the high expansion case.

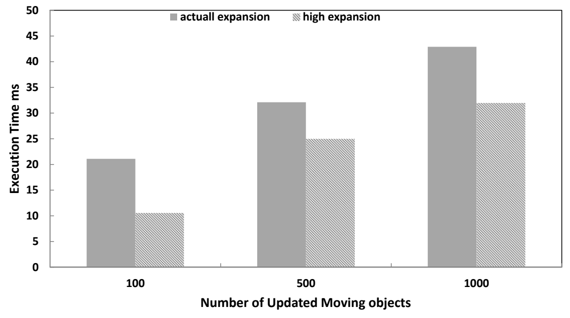


Fig. 23 The effect of expansion complexity on updating objects for different densities

Summary. In our evaluation experiments, we evaluated the existing parameters in our indoor tree in order to test the proposed indoor index. We examined the unique parameters in our data structure in order to determine the effect on the data structure (e.g. moving objects number, connection complexity, expansion complexity and cells density). We summaries the results as follows: Three important observations follow from the conducted experiments. First, the effect of increasing the number of moving objects and the increasing the complexity of cell connections (high adjacency) have no explicit influence on the tree constructions. Second, the query costs and the insertion costs perform efficiently with different cases having different numbers of moving objects, cells connections and the expand points. Third, the indoor-tree can successfully obtain a reliable and a robust tree index based on the novel idea of indoor connections and adjacency.

6 Conclusion and Future Work

This paper addresses the challenge of building an index data structure that is appropriate for indoor spaces. The measurement of indoor space is different from that of outdoor space that is based on Euclidean space or a spatial network. Indoor space is related to the notion of cellular space that contains different entities such as rooms, doors and hallways that enable or constrain movement. The proposed index takes into account the entities that enable or constrain the movement in indoor environment (doors and hallways). Our tree structure is based on adjacency and cell connections, which allows us to answer spatial queries more efficiently. In addition, our data structure is based also on the indoor expansion, which is the way that the indoor floor is divided from its main entrance to the last cell. This enables us to answer future queries more efficiently. The expansion algorithm enables us to use the expand points and record the two expand points with non-leaf nodes as range, and the largest expand point will be recorded in the leaf nodes (for comparison and to choose nodes' goals). To the best of our knowledge, using expand points in choose *ChooseLeaf* comparison is unique in our tree data structure. Extensive performance studies were conducted and results indicate that the indoor-tree is both robust and efficient for the targeted queries. In fact, the query costs and the insertion costs perform efficiently with different cases such as moving objects number, cells connections number and the expand points number, which indicate that our indoor tree is reliable and robust.

Several directions can be extended from this work. First, serving new types of queries such as the temporal queries and the trajectories based on the adjacency method. The temporal aspects are one of the challenges that we intend to investigate for the indoor space index. Moreover, we aim to extend our indoor tree to include different spatial queries and different navigational queries. Another avenue of research is the investigation of the data structure of moving objects with new patterns of movement within the indoor spaces.

Acknowledgements The source of our implementations can be downloaded from the following URL: `http://users.monash.edu/~sultana/indoortree.rar`

References

1. Arne Andersson, Torben Hagerup, Johan Håstad, and Ola Petersson. The complexity of searching a sorted array of strings. In *Proceedings of the Twenty-sixth Annual ACM Symposium on Theory of Computing*, STOC '94, pages 317–325, New York, NY, USA, 1994.
2. ChaeHoon Ban, BongHee Hong, and DongHyun Kim. Time parameterized interval r-tree for tracing tags in rfid systems. In Kim Andersen, John Debenham, and Roland Wagner, editors, *Database and Expert Systems Applications*, volume 3588 of *Lecture Notes in Computer Science*, pages 503–513. Springer Berlin / Heidelberg, 2005.
3. V. Prasad Chakka, V. Prasad, Chakka Adam, Adam C. Everspaugh, and Jignesh M. Patel. Indexing large trajectory data sets with seti, 2003.
4. Jae-Woo Chang, Jung-Ho Um, and Wang-Chien LeeP. A new trajectory indexing scheme for moving objects on road networks. In David Bell and Jun Hong, editors, *Flexible and Efficient Information Handling*, volume 4042 of *Lecture Notes in Computer Science*, pages 291–294. Springer Berlin / Heidelberg, 2006.
5. Yong-Jin Choi, Jun-Ki Min, and Chin-Wan Chung. A cost model for spatio-temporal queries using the TPR-tree. *Journal of Systems and Software*, 73(1):101 – 112, 2004.
6. Ying Fang, Jiaheng Cao, Junzhou Wang, Yuwei Peng, and Wei Song. Htpr*-tree: An efficient index for moving objects to support predictive query and partial history query. In Liwei Wang, Jingjue Jiang, Jiaheng Lu, Liang Hong, and Bin Liu, editors, *Web-Age Information Management*, volume 7142 of *Lecture Notes in Computer Science*, pages 26–39. Springer Berlin Heidelberg, 2012.
7. F. Forno, G. Malnati, and G. Portelli. Design and implementation of a bluetooth ad hoc network for indoor positioning. *Software, IEE Proceedings* -, 152(5):223 – 228, oct. 2005.
8. Elias Frentzos. Indexing objects moving on fixed networks. In *In Proc. of the 8th Intl. Symp. on Spatial and Temporal Databases (SSTD*, pages 289–305. Springer, 2003.
9. Alan Gibbons. *Algorithmic Graph Theory.* Cambridge University Press, 1985.
10. Ralf Hartmut Güting, Michael H. Böhlen, Martin Erwig, Christian S. Jensen, Nikos A. Lorentzos, Markus Schneider, and Michalis Vazirgiannis. A foundation for representing and querying moving objects. *ACM Trans. Database Syst.*, 25(1):1–42, March 2000.
11. Antonin Guttman. R-trees: A dynamic index structure for spatial searching. In *International Conference on Management of Data*, pages 47–57, 1984.

12. Hui-Huang Hsu and Chien-Chen Chen. Rfid-based human behavior modeling and anomaly detection for elderly care. *Mobile Information Systems*, 6(4):341–354, 2010.
13. Haibo Hu, Dik Lun Lee, and Victor C. S. Lee. Distance indexing on road networks. In *Proceedings of the 32nd International conference on Very Large Data Bases*, VLDB '06, pages 894–905. VLDB Endowment, 2006.
14. Christian Jensen, Hua Lu, and Bin Yang. Indexing the trajectories of moving objects in symbolic indoor space. In Nikos Mamoulis, Thomas Seidl, Torben Pedersen, Kristian Torp, and Ira Assent, editors, *Advances in Spatial and Temporal Databases*, volume 5644 of *Lecture Notes in Computer Science*, pages 208–227. Springer Berlin / Heidelberg, 2009.
15. Jay Yellen Jonathan L. Gross. *Graph Theory and Its Applications*, pages 585–655. Chapman and Hall/CRC, 2005.
16. Ibrahim Kamel and Christos Faloutsos. Hilbert r-tree: An improved r-tree using fractals. In *Proceedings of the 20th International Conference on Very Large Data Bases*, VLDB '94, pages 500–509, San Francisco, CA, USA, 1994.
17. Hye-Young Kang, Joon-Seok Kim, and Ki-Joune Li. Similarity measures for trajectory of moving objects in cellular space. In *Proceedings of the 2009 ACM Symposium on Applied Computing*, SAC '09, pages 1325–1330, New York, NY, USA, 2009.
18. Yongjun Li, Hu Chen, Rong Xie, and James Z. Wang. Bgn: A novel scatternet formation algorithm for bluetooth-based sensor networks. *Mobile Information Systems*, 7(2):93–106, 2011.
19. Wei Liao, Guifen Tang, Ning Jing, and Zhinong Zhong. Vtpr-tree: An efficient indexing method for moving objects with frequent updates. In *Advances in Conceptual Modeling - Theory and Practice*, volume 4231 of *Lecture Notes in Computer Science*, pages 120–129. Springer Berlin / Heidelberg, 2006.
20. Bin Lin and Jianwen Su. On bulk loading TPR-tree. In *Proceedings of the International Conference on Mobile Data Management*, pages 114 – 124, 2004.
21. Dan Lin. *Indexing and Querying Moving Objects Databases*. PhD thesis, National University of Singapore, Singapore, 2006.
22. Dan Lin, Rui Zhang, and Aoying Zhou. Indexing fast moving objects for knn queries based on nearest landmarks. *Geoinformatica*, 10(4):423–445, December 2006.
23. Mario A. Nascimento and Jefferson R. O. Silva. Towards historical r-trees. In *Proceedings of the 1998 ACM Symposium on Applied Computing*, SAC '98, pages 235–240, New York, NY, USA, 1998.
24. Atsuyuki Okabe, Barry Boots, Kokichi Sugihara, Sung Nok Chiu, and D. G. Kendall. *Spatial Tessellations:Concepts and Applications of Voronoi Diagrams*, pages 585–655. John Wiley Sons, Inc., 2008.
25. Mario Muñoz Organero, Pedro J. Muñoz Merino, and Carlos Delgado Kloos. Using bluetooth to implement a pervasive indoor positioning system with minimal requirements at the application level. *Mobile Information Systems*, 8(1):73–82, 2012.
26. Dieter Pfoser, Christian S. Jensen, and Yannis Theodoridis. Novel approaches to the indexing of moving object trajectories. In *International Conference on Very Large Databases*, pages 395–406, 2000.
27. Toms Ruiz-Lpez, Jos Garrido, Kawtar Benghazi, and Lawrence Chung. A survey on indoor positioning systems: Foreseeing a quality design. In Andre de Leon F. de Carvalho, Sara Rodrguez-Gonzlez, Juan De Paz Santana, and Juan Rodrguez, editors, *Distributed Computing and Artificial Intelligence*, volume 79 of *Advances in Intelligent and Soft Computing*, pages 373–380. Springer Berlin / Heidelberg, 2010.
28. Simonas Saltenis, Christian S. Jensen, Scott T. Leutenegger, and Mario A. Lopez. Indexing the positions of continuously moving objects. *SIGMOD Rec.*, 29(2):331–342, May 2000.
29. Yufei Tao and Dimitris Papadias. Mv3r-tree: A spatio-temporal access method for timestamp and interval queries. In *Proceedings of the 27th International Conference on Very Large Data Bases*, VLDB '01, pages 431–440, San Francisco, CA, USA, 2001.
30. Yufei Tao, Dimitris Papadias, and Jimeng Sun. The TPR*-tree: An optimized spatio-temporal access method for predictive queries. In *In VLDB*, pages 790–801, 2003.
31. Yannis Theodoridis, Michael Vazirgiannis . Timos Sellis, Michael Vazirgiannis, and Timos Sellis. Spatio-temporal indexing for large multimedia applications. In *ICMCS*, pages 441–448, 1996.
32. Agustinus Borgy Waluyo, Bala Srinivasan, and David Taniar. Research in mobile database query optimization and processing. *Mobile Information Systems*, 1(4):225–252, December 2005.
33. O. Wolfson, B. Xu, S. Chamberlain, and L. Jiang. Moving objects databases: issues and solutions. In *Scientific and Statistical Database Management, 1998. Proceedings. Tenth International Conference on*, pages 111 –122, jul 1998.
34. Wenjie Yuan and Markus Schneider. Supporting continuous range queries in indoor space. In *Proceedings of the 2010 Eleventh International Conference on Mobile Data Management*, MDM '10, pages 209–214, Washington, DC, USA, 2010. IEEE Computer Society.
35. Geng Zhao, Kefeng Xuan, W. Rahayu, D. Taniar, M. Safar, M.L. Gavrilova, and B. Srinivasan. Voronoi-based continuous k nearest neighbor search in mobile navigation. *IEEE Transactions on Industrial Electronics*, 58(6):2247 –2257, june 2011.