Pers Ubiquit Comput (2014) 18:287–301
DOI 10.1007/s00779-013-0645-3

ORIGINAL ARTICLE

A connectivity index for moving objects in an indoor cellular space

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

Received: 8 October 2012 / Accepted: 31 January 2013 / Published online: 18 April 2013

Abstract With the currently available indoor positioning devices such as RFID, Bluetooth and WI-FI, the locations of moving objects constitute an important foundation for a variety of applications such as the tracking of 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 is based on the connectivity (adjacency) between the indoor environment cells and 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 [13, 17]. 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, the positioning and tracking of moving objects is an important research field with many applications including the tracking of moving objects, way finding and security [24].

Current global navigation satellite systems such as GPS are not suitable for indoor environments (spaces). With the new technologies for positioning devices for indoor spaces such as RFID, Bluetooth and Wi-Fi [6], 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 such as Wi-Fi have become more advanced [22, 24]. 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 [13, 17, 23] and the indexing and querying of the trajectories of moving objects and their locations [3, 5, 18, 20, 25]. 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 applied to 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 an 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 an outdoor space, a GPS is capable of

S. Alamri (✉) · D. Taniar · H. Al-Khalidi
Clayton School of Information Technology, Monash University, Clayton, Australia
e-mail: Sultan.Alamri@monash.edu

D. Taniar
e-mail: david.taniar@monash.edu

H. Al-Khalidi
e-mail: Haidar.Al-Khalidi@monash.edu

M. Safar
Computer Engineering, Kuwait University, Safat, Kuwait
e-mail: maytham.safar@ku.edu.kw

continuously reporting the velocity and location of a moving object with varying accuracy. 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 of objects [9, 13, 30]. Examples of indoor positioning devices are the RFID reader and Bluetooth which are based on the sensing or activation range of a positioning device.

With the new technologies of the smart phones, many indoor applications have become available. Tracking the mobile users has become easy. Applications that assist the mobile users to determine a nearest moving objects to a certain point in indoor floor. Also security applications which assist to determine which object checked in or checked out from a certain room. Motivated by that, the indexing of the moving objects in the indoor spaces has become more needed.

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 behind 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 terms 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 [13, 32]. 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 neighbors' distance lookup table and efficiently 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 building 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 [3, 4, 23]: We start with the Trajectory Bundle tree (TB-tree), which is based on the R-tree [10, 21]. 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, 17]. Another trajectory index is named Spatio-Temporal R-tree (STR-tree) [3, 4, 23]. STR is based not only on spatial closeness, but also on 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 the same trajectory [23]. The Scalable and Efficient trajectory Index (SETI) [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 are indexed by a separate spatial index (R-tree) [3, 13].

Works that concentrate on *historical* moving objects [20, 26, 28]: 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 to concentrate on historical data [20]. 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 [20, 26]. Another work that focuses on the historical data are the Multi-version 3D R-tree (MV3R-tree) [26] which basically uses Multi-version B-trees [17, 21] and combines them with 3D R-trees [28]. The MV3R-tree includes significant improvements which produce huge space savings without influencing the performance of the timestamp queries compared to the HR-tree. Another work that focuses on the historical data but for a road network is the Fixed Network R-tree (FNR-tree) proposed by Frentzos [7]. The idea is to take into account the constraints in road networks. The FNR-tree contains a two-dimensional (2D) R-tree and a one-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 [5, 18, 25]: Saltenis et al. introduced the Time Parameterized R-tree, (TPR-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 minimum bounding rectangle (MBR), beside its velocity vector which they call VBR. As an extension of the TPR-Tree, Yufei Tao proposed the TPR*-tree [27] 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. Our DV-TPR*-Tree provides an efficient data structure for the moving objects based on the direction and the velocity [1]. Moreover, some works such as [19] 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 [9]. 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 or 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 devices' sensing ranges might cause large volatility of the data if outdoor data structures are used [6, 13, 30].

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 object's location based on its coordinate on the floor. We assume that the objects are indexed by a metric data structure such as (TPR-tree). Moreover, the system is dynamic; that is, objects may be deleted or new objects may be inserted. Table 1 explains the notations used throughout this paper.

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 Fig. 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

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 of 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
$Ci \rightarrow Cj$	Ci continue the movement from the DC toward the last cell
$Ci(N\chi)$	The number of connections in Ci
MIN	Minimum values
RC	Range of expand points (cells)
LE	Largest expand point
DC	Default cell
N	Leaf node
$EX\text{-}P$	*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

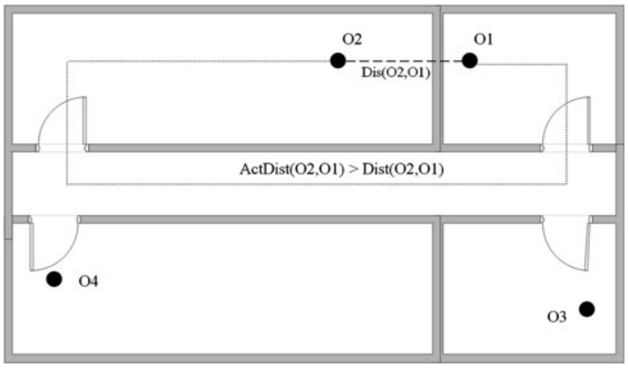


Fig. 1 Using *R*(*P*) in indoor space can return wrong results; in this example, *O*2 is chosen as the 1stNN of *O*1, where the right result is *O*3 because of *ActDist*(*O*1, *O*3) < *ActDist*(*O*1, *O*2)

*O*2 = [*xj*, *yj*] , *O*3 = [*xr*, *yr*] and *O*4 = [*xe*, *ye*]. Based on the metric properties of the object's structure the result will be *O*2. However, we can notice that *O*2 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*(*O*1, *O*2) > *Dist* (*O*1, *O*2)), *O*2 is the farthest one from *O*1, and *O*3 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 in order to obtain the accurate 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 notion of cell connectivity [where the number of hops between the cells is used instead of Euclidean distance, (see Fig. 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.)

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

Definition 2 Given a set of moving objects $P = \{O1, O2, \ldots, On\}$, If *CU*(*P*), the *CellDist*(*Ox*, *Oy*) is the number of hops of the cells, and calculated as: $CellDist(Ox, Oy) = |\{C1, C2, \ldots, Cn\} - 1|$, where $Ox \overset{in}{\rightarrow} C1$ and $Oy \overset{in}{\rightarrow} Cn$, $\forall C1 \chi C2 \chi C8 \ldots \chi Cn$.

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

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

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 according to sets of cells that include spatial objects. The indoor space

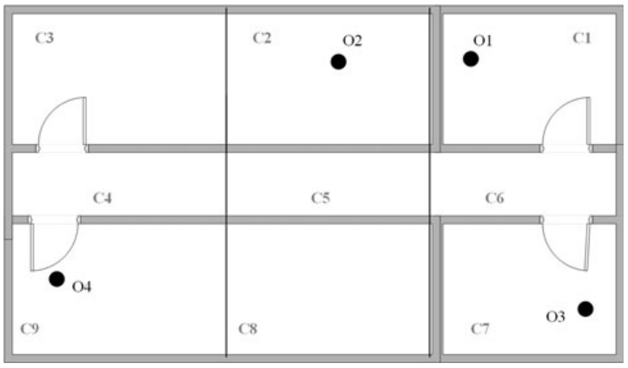


Fig. 2 An example of the cells' coverage and distribution which is more likely to return an accurate result based on the neighbors and connections between the cells

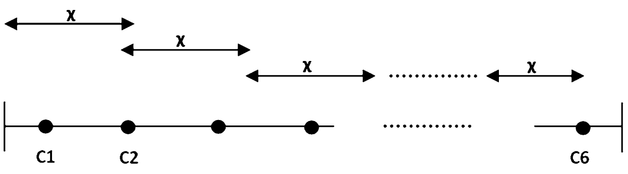


Fig. 3 The number of cells hops from *C*1 to *C*6 *CellDist*(*O*1,*O*6)

data structure has an important requirement related to the notion of cellular space.

Definition 4 Given a set of cells $C = \{C1, C2, \ldots, Cn\}$, space S and a set of spatial objects $\{O1, O2, \ldots, On\}$, 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 $C = \{C1, C2, \ldots, Cn\}$, 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 [13, 30, 33]. 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 the 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 [9, 11, 15].

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 [12, 15, 22]. For more explanation, in a geometric space, both the located objects and locations are shown as coordinate n-tuples, represented as points, areas and volumes [16, 17, 29]. 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 cellular space.

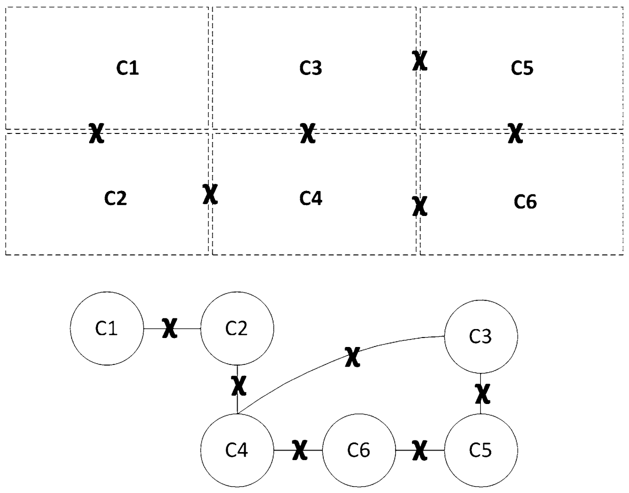


Fig. 4 An example of the coverage cells distribution, where χ means connected

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 is divided into grids based on the coverage provided by its positioning devices (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 gives an example of the coverage cells distribution.

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

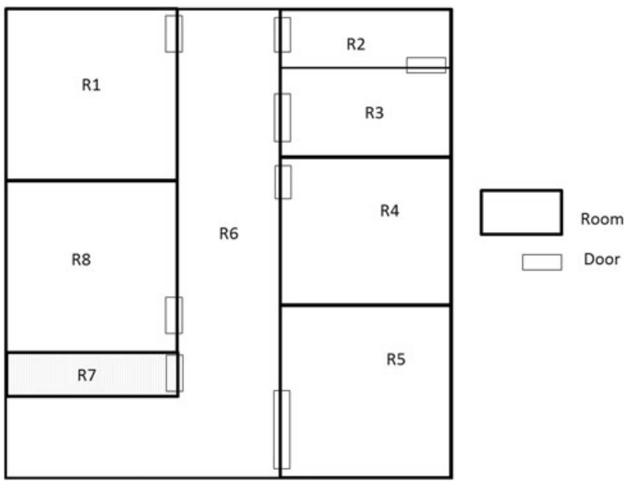


Fig. 5 An example of floor plane of 7 rooms

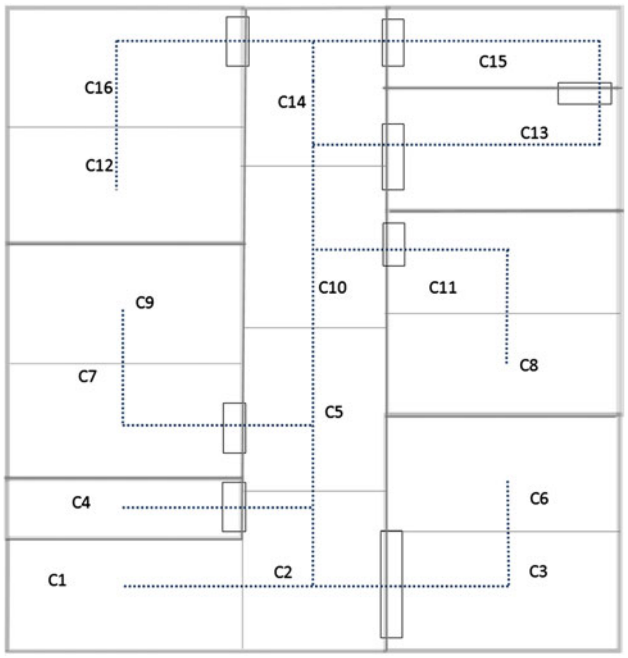


Fig. 6 Illustration of the connectivity between the cells

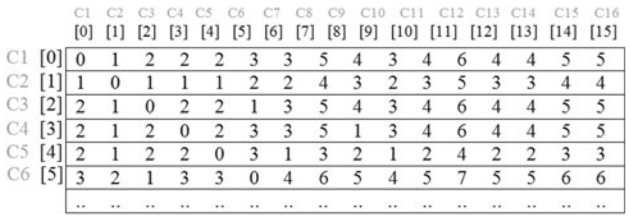
coverages is addressed by creating a new individual cell for any cells overlap. Since the issue of cells overlapping in indoor spaces is a considerable research area, we will leave this detail for future work.

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 (Fig. 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.

The connection between the cells is stored in a neighbors' distance lookup table to facilitate the grouping in the data structure. In this table, we have pre-computed neighbors' distances between cells. The distance is not metric, but is based on the number of hops, where 0 means the cell itself, 1 means neighbor number one, 2 means neighbor number two and so on. We assume that the positioning devices' coverage and the neighbors' distance lookup are pre-fixed. The neighbors' 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 in order to obtain the adjacent cells in case of inserting, deleting or updating (to determine the suitable nodes). Note that the size of the neighbors' distance lookup table is small (Fig. 7). Moreover, we store the cells' connection as a multi-dimensional array list where the search complexity is $O(n)$ [2]. 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 the neighbors' distances lookup table).

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
```



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

4.1 Indoor filling space representation

Indoor space is a filling space based on the connection between the cells [9, 15]. 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 neighbors' 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 on the indoor floor.

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

Definition 8 Given a set of cells $C = \{C1, C2, \ldots, Cn\}$, the expand points' order are $C1 > C2, \ldots, > Cj$, iff:

- $C1$ is the *DC* and Cn is the last cell, and
- $C1 \rightarrow C2,\ C2 \rightarrow Cx, \ldots, Cj \rightarrow Cn,\ \forall\ j \neq x$ and $x \neq n$.

Since the cells in the indoor space overlap, as shown in Fig. 4, the cells connection is represented as an acyclic graph; from the default cell *DC*, we convert (based on

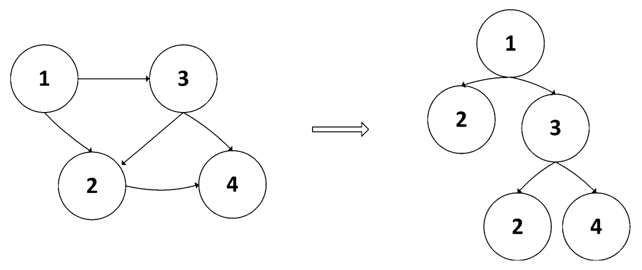


Fig. 8 Converting acyclic graph to tree

definition 7, 8) the acyclic graph to tree [8, 31]. Figure 8 shows as example of a tree acyclic graph converted to tree. The indoor expansion will be explained next.

Our objective is to use the data of the expansion idea (stored similarly to the neighbors' 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 take advantage of this and record the two expand points with non-leaf nodes as range and the largest expand point is recorded in the leaf nodes (for comparison and to choose node goals). The steps for the indoor filling space (expansion) algorithm are as follows:

- The algorithm establishes *expand points* which are the points (cells) based on definition 7, $C1 \rightarrow C2$, $C2 \rightarrow Cj, \ldots, Cj \rightarrow Cn$.
- The expansion starts 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. For example based on Fig. 9, $C4(N\chi) < C3(N\chi)$.
- Then the algorithm will list the cells that have more adjacent cells (more connections); for example, based on Fig. 9, $C5(N\chi) > C3(N\chi)$.

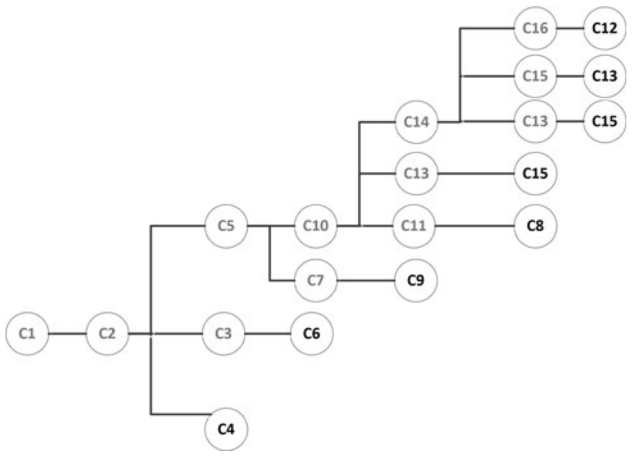


Fig. 9 The expand steps of the floor example (*gray* the expand points)

- The process is repeated up to the *last cell* (each cell that continues the connection will be considered as an *EX-P*).

Referring to Fig. 9, assuming that *C*1 is the default cell, the algorithm will start from *C*1 and assign it as an *EX-P*. Then this will expand to the adjacent cells starting from the cells that have fewer connections (*C*4) to the cells that have more connections (*C*3 and *C*5). Cells that continue the connection will be assigned as expand points (e.g., *C*3 and *C*5). 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 (*C*1)) to the last cell on the floor (in our case *C*12). 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 a better understanding of 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 [10]. 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 *Largest Hilbert Value* (*LHV*) [14]. 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 groups the entries based on their adjacency cells. This is the best means 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 range cells (*RC*) 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 $C = \{C1, C2, \ldots, Cn\}$, and set of non-leaf nodes $N = \{N1, N2, \ldots, Nn\}$, the *RC (Range cells)* in *Ni* is the two expand cells as follows:

- $\lfloor Ci \rfloor$ is highest expand point and $\lceil Cj \rceil$ is lowest expand point, where $Ci > Cx, \ldots, > Cj$, $\forall Ci, Cx, \ldots, Cj \in Ni$.

Definition 10 Given a set of cells $C = \{C1, C2, \ldots, Cn\}$, and set of leaf nodes $N = \{N1, N2, \ldots, Nn\}$, the *LE* is highest expand point $\lfloor Ci \rfloor$, where $Ci > Cx, \ldots, > Cj$, $\forall\ Ci, Cx, \ldots, Cj \in Ni$.

Example For the non-leaf nodes, using the data in Fig. 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 *N*1 is (*C*10;*C*16)(where *C*10 is the highest expand cell contained in *N*1 and *C*16 is the lowest). For the leaf nodes, using the data in Fig. 12, the leaf node $N1a$ has two cells = $\{C12, C16\}$. The *LE* for $N1a$ is (*C*10)(where *C*10 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 (gray in Fig. 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 is 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*, and *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 Fig. 10.

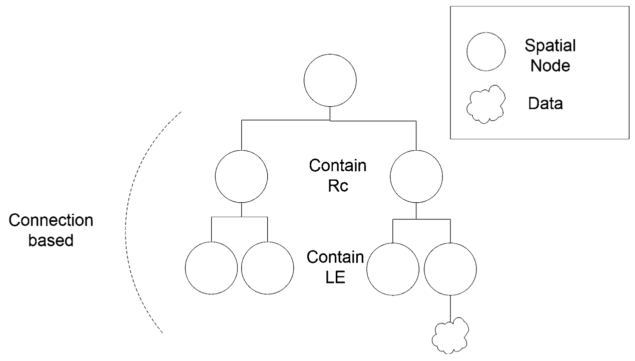


Fig. 10 Indoor-tree

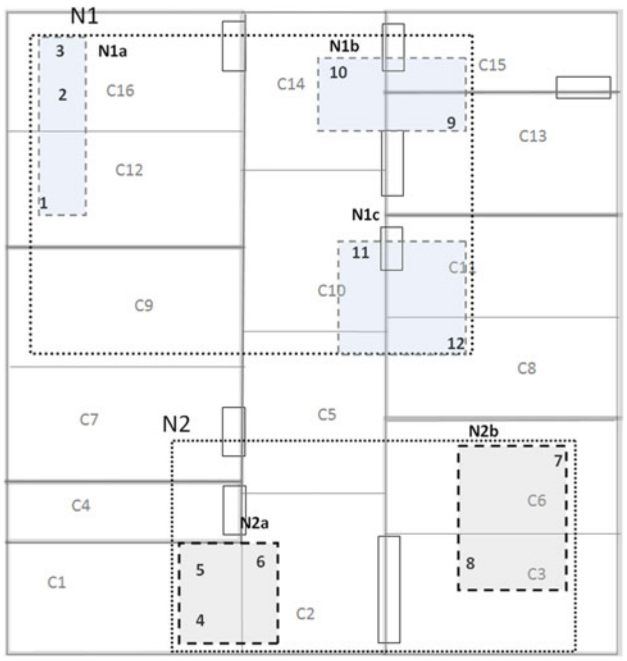


Fig. 11 The MBRs grouping based on the connection

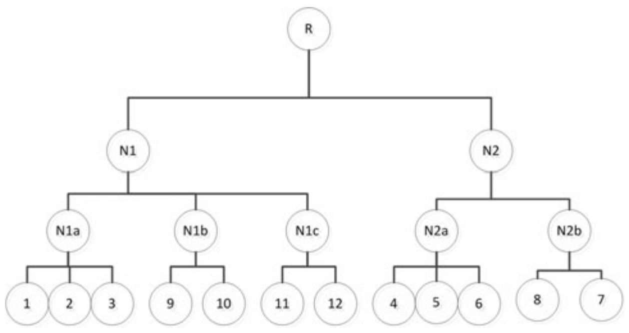


Fig. 12 An example of an indoor-tree

Using the sample data in Fig. 11, suppose that the five MBRs are clustered into a larger MBR, where moving objects $= 1, 2, 3, 4, \ldots, 12$ 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 ($C10$), $N2a$ *LE* is ($C1$) and $N2b$ *LE* is ($C3$). The indoor-tree is shown in Fig. 12.

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 needs to be grouped with the adjacent cells.

Definition 11 Given a set of Nodes $N = \{N1, \ldots, Nn\}$, and spatial objects $O1, O2, \ldots, On$, 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 $N = \{N1, \ldots, Nn\}$, and spatial objects $O1, O2, \ldots, On$, 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 checks the inserted cell with the *LE* and choose the node that has the MIN value cell. For example, using the sample data in Fig. 11, assume that an entry '13' will be inserted to $C16$. The algorithm starts 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 Fig. 13. If the *RC* comparison results and LE comparison results are equal, the algorithm will choose the node that has more neighbors (connections) (algorithm 2).

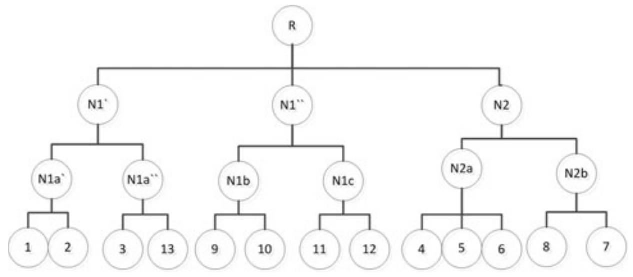


Fig. 13 Inserting 13 to $N1a''$

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
```

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
```

In the deletion operation, we check for the underflowing node, instead of the overflowing node. If a node is underflowing, we need to check the sibling nodes that are based on the connection to participate in solving the underflow. The deletion algorithm will be explained in the next example. Suppose that we want to delete object 7 from *C*6 where $m = 2$ and $M = 3$ (Fig. 12). The algorithm will first call the FindLeaf algorithm which iteratively searches for the target node from the root node to the leaf node [10, 27]. After discovering that *N*2*b* is underflown, the delete algorithm will determine the sibling-connected node based on the adjacency comparison algorithm (Fig. 14). Then re-insert the remain entities into that node (if it is overflowing we split, as explained in the insertion algorithm). Hence, in the deletion, if underflow occurs, we deal with the adjacent cells.

Algorithm 5 update algorithm

```
 1: /* Input: $o$ is the entry to be updated. */
 2: procedure UPDATE($o$ from $C$)
 3:     FindLeaf to find the leaf node $N$
 4: // the FindLeaf function
 5:     if $o$ move to adjacent cells then
 6:         positioning device skip updating
 7:     end if
 8:     if $N$ effected then
 9:         if $N$ underflows then
10:             ChooseLeaf function to find $N$ sibling adjacent
11:             re-insert $o$ into $N_i$
12:         end if
13:         if $N_i$ overflows then
14:             HandleOverflow($N_i, o$)
15:         end if
16:     end if
17:     if node split propagated to the root node and root node to split then
18:         Create a new root node with new resulting child nodes
19:     end if
20: end procedure
```

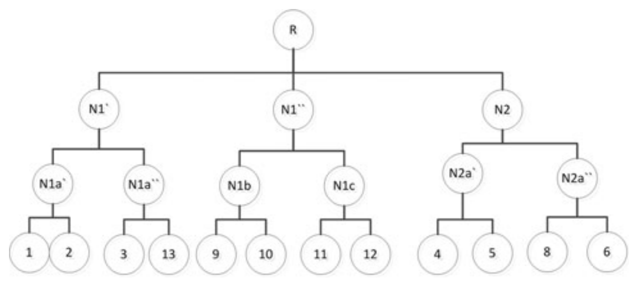


Fig. 14 After deleting 7, object 8 is reinserted to *N2a*

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 as in the original R-tree [10]. 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 one cell to another without any updating in the nodes (since a node contains the adjacent cells).

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 2.50 GHz 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 dataset size ranges from 10 to 1,000 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 an indoor space floor. We generate the location of the

Table 2 Parameters and their settings

Parameter	Setting
Node capacity	60
Number of moving objects	(10–1,000)
Indoor space	12, 50, 100, 150 cells
Expand cells	Increase with the increasing of the indoor cells
Operations	Insert, find
Dataset	Synthetic

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 the number of 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 a 12-cells real-case 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 measure the costs for the different densities of moving objects (number of the moving object) and the indoor space overlapping or the connection complexity. For the query performance and maintaining operations test, we 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 times and the average is calculated. The parameters used are summarized in Table 2.

5.1 Tree construction

Tree construction costs are illustrated in Figs. 15 and 16. We generate moving objects ranging from 10 to 1,000 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 in the moving objects for each indoor cells case. As the number of moving objects increases, the construction cost increases accordingly. This behavior is natural in an indexing tree where, with the increase in the moving objects, the construction cost will increase, and the number of the nodes and splits will increase; moreover, the comparison and

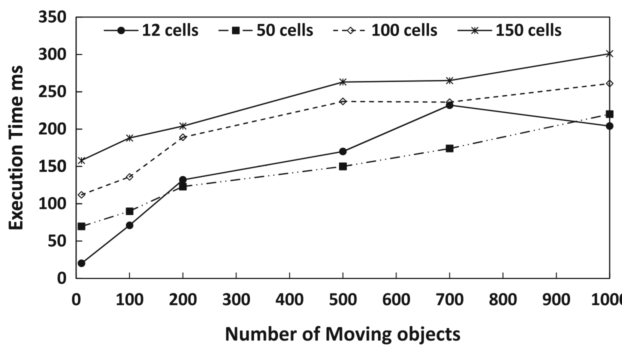


Fig. 15 Effect of objects number

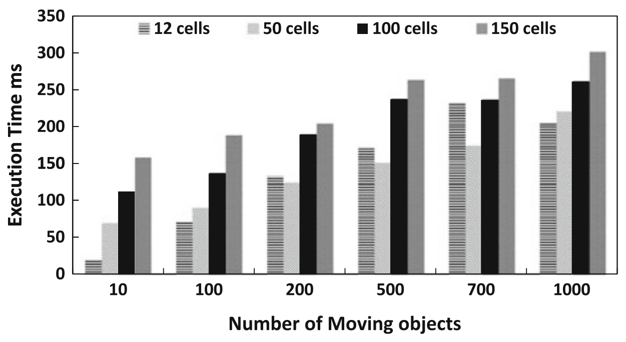


Fig. 16 Effect of cells numbers

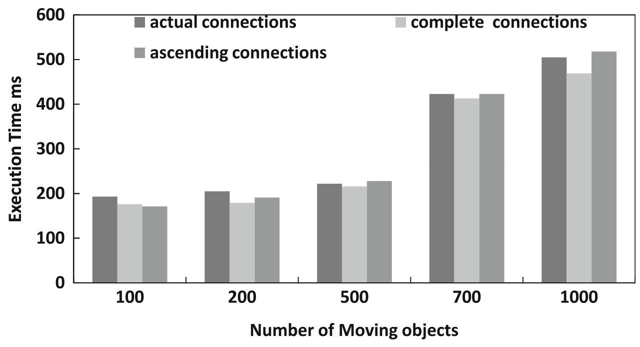


Fig. 17 Illustrates the effect of connection complexity

checking of the lookup table for the MIN values will be increased. Figure 16 illustrates the increase in the number of the cells. We can see 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 shows the effect of the cell connections' complexity. Basically, we tested the construction on different complexities of connections. Apart from the actual connection on the floor, we include the worst case connection which is a complete graph connection, and the best case is ascending connections [8]. Note that the increase in the complexity of the cells' connections does not impact on the tree construction costs. We can explain this behavior as follows: when we insert any objects into any cell, with the increase in 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

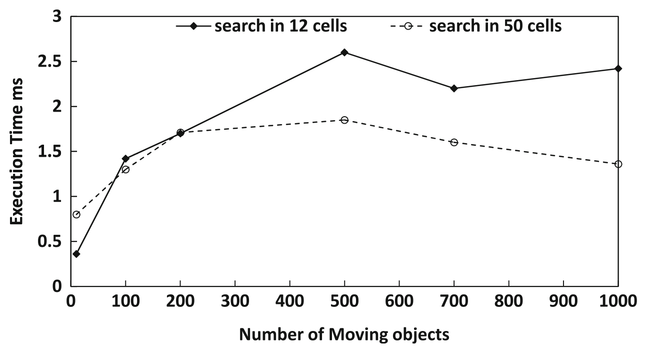


Fig. 18 Effect of objects densities (search performance)

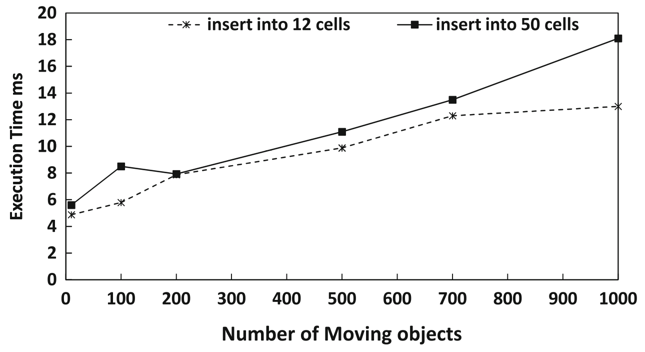


Fig. 19 Effect of objects densities (insert performance)

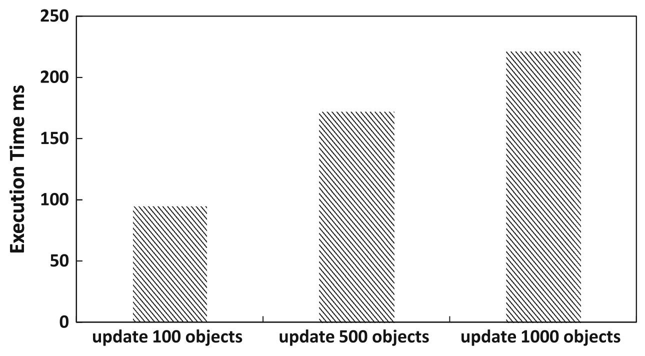


Fig. 20 Illustrates the effect of different updated objects densities

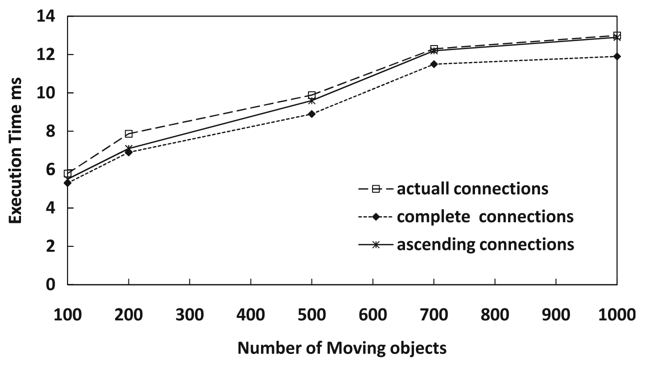


Fig. 21 The effect of connection complexity on inserting

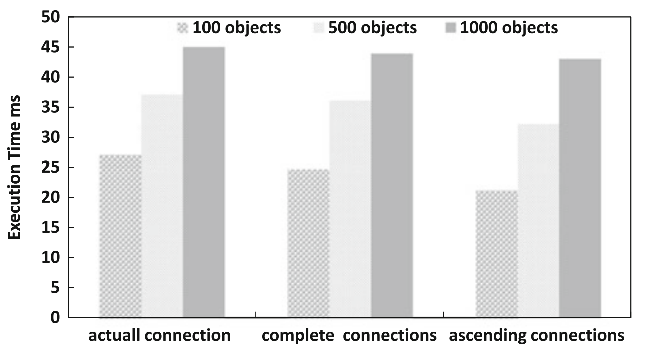


Fig. 22 The effect of connection complexity on updating objects on different densities

proposed insert algorithm with its associated operations cost. Moreover, we consider the effects of increasing 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 conducted after constructing the moving objects, where we search for particular objects in object densities of 10, 100, 200, 500, 700 and 1,000. 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 is used.

5.2.1 *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 Fig. 18 that with the increase in the number of moving objects, the query cost increases slightly around 1.4 ms. Moreover, we can notice that for the 12-cells case, when we perform a query at 1,000 density, the cost is almost similar to that for other densities. Note that the search query cost is usually less than the insertion cost, 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 10–1,000 objects, and then we record the average time for each moving objects' density. We can see from Fig. 19 that in some cases, with the increase in the number of moving objects, the cost increases around 8 ms. From Fig. 19, we can notice that the insertion cost from the 12-cells case, increases slightly with the increase in the number of moving objects, due to the increase in the related operations and nodes. However, we can also see that for a density of 500 objects in 12 cells, the performance is close to that of the 500 objects density for 50 cells.

Furthermore, Fig. 20 illustrates the updating of the indoor-tree of 50 cells at density 1,000 objects. We update 100 objects first, then 500 objects, then all moving objects. We note that our indoor-tree still performs efficiently when updating large densities of moving objects.

5.2.2 *Effect of connections number*

Here, we measure the complexity based on the cells' connections, which means that we increase the cells' neighbors 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. 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 Fig. 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 neighbors. Moreover, the splitting can also be facilitated in environments with more connections, where any node has to be split (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. Here we can see that the indoor-tree still performs well in the worst case connection.

5.2.3 *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.

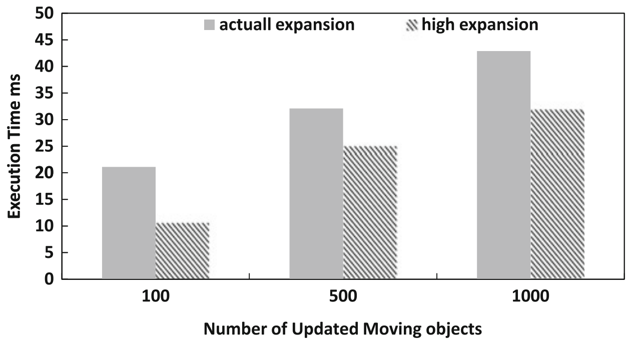


Fig. 23 The effect of expansion complexity on updating objects for different densities

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

5.2.4 *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 summarize the results as follows: Three important observations follow from the conducted experiments. First, increasing the number of moving objects and increasing the complexity of cell connections (high adjacency) have no explicit effect 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 an 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 is recorded in the leaf nodes (for comparison and to choose nodes' goals). To the best of our knowledge, using expand points in *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, is work related to 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.

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

References

1. Alamri S, Taniar D, Safar M (2012) Indexing moving objects for directions and velocities queries. Inf Syst Front 1–14. doi:10.1007/s10796-012-9367-8
2. Andersson A, Hagerup T, Håstad J, Petersson O (1994) The complexity of searching a sorted array of strings. In: Proceedings of the twenty-sixth annual ACM symposium on theory of computing, STOC '94, pp 317–325, New York, NY, USA
3. Chakka VP, Everspaugh A, Patel JM (2003) Indexing large trajectory data sets with *SETI*. In: CIDR. http://www-db.cs.wisc.edu/cidr/cidr2003/program/p15.pdf
4. Chang J-W, Um J-H, LeeP W-C (2006) A new trajectory indexing scheme for moving objects on road networks. In: Bell D, Hong J (eds) Flexible and efficient information handling. Lecture notes in computer science, vol 4042. Springer, Berlin, pp 291–294
5. Choi Y-J, Min J-K, Chung C-W (2004) A cost model for spatio-temporal queries using the TPR-tree. J Syst Softw 73(1):101–112
6. Forno F, Malnati G, Portelli G (2005) Design and implementation of a Bluetooth ad hoc network for indoor positioning. IEE Proc Softw 152(5):223–228
7. Frentzos E (2003) Indexing objects moving on fixed networks. In: Proceedings of the 8th international symposium on spatial and temporal databases (SSTD), Springer, pp 289–305
8. Gibbons A (1985) Algorithmic graph theory. Cambridge University Press, Cambridge
9. Güting RH, Böhlen MH, Erwig M, Jensen CS, Lorentzos NA, Schneider M, Vazirgiannis M (2000) A foundation for representing and querying moving objects. ACM Trans Database Syst 25(1):1–42
10. Guttman A (1984) R-trees: a dynamic index structure for spatial searching. In: International conference on management of data, pp 47–57
11. Hsu H-H, Chen C-C (2010) Rfid-based human behavior modeling and anomaly detection for elderly care. Mobile Inf Syst 6(4): 341–354
12. Hu H, Lee DL, Lee VCS (2006) Distance indexing on road networks. In: Proceedings of the 32nd international conference on very large data bases, VLDB '06. VLDB Endowment, pp 894–905
13. Jensen C, Lu H, Yang B (2009) Indexing the trajectories of moving objects in symbolic indoor space. In: Nikos M, Thomas S, Torben P, Kristian T, Ira A (eds) Advances in spatial and temporal databases. Lecture notes in computer science, vol 5644. Springer, Berlin, pp 208–227
14. Kamel I , Faloutsos C (1994) Hilbert r-tree: An improved r-tree using fractals. In: Proceedings of the 20th international conference on very large data bases, VLDB '94, pp 500–509, San Francisco, CA, USA
15. Kang H-Y, Kim J-S, Li K-J (2009) Similarity measures for trajectory of moving objects in cellular space. In: Proceedings of the 2009 ACM symposium on applied computing, SAC '09, pp 1325–1330, New York, NY, USA
16. Li Y, Chen H, Xie R, Wang JZ (2011) Bgn: a novel scatternet formation algorithm for bluetooth-based sensor networks. Mobile Inf Syst 7(2):93–106
17. Lin D (2006) Indexing and querying moving objects databases. PhD thesis, National University of Singapore, Singapore
18. Lin B, Su J (2004) On bulk loading TPR-tree. In: Proceedings of the international conference on mobile data management, pp 114–124
19. Lin D, Zhang R, Zhou A (2006) Indexing fast moving objects for knn queries based on nearest landmarks. Geoinformatica 10(4):423–445
20. Nascimento MA, Silva JRO (1998) Towards historical R-trees. In: Proceedings of the 1998 ACM symposium on applied computing, SAC '98, pp 235–240, New York, NY, USA
21. Okabe A, Boots B, Sugihara K, Chiu SN, Kendall DG (2008) Spatial tessellations: concepts and applications of Voronoi diagrams. In: Spatial tessellations. Wiley, pp 585–655. doi:10.1002/9780470317013.refs
22. Organero MM, Merino PJM, Kloos CD (2012) Using Bluetooth to implement a pervasive indoor positioning system with minimal requirements at the application level. Mobile Inf Syst 8(1):73–82
23. Pfoser D, Jensen CS, Theodoridis Y (2000) Novel approaches to the indexing of moving object trajectories. In: International conference on very large databases, pp 395–406
24. Ruiz-Lpez T, Garrido J, Benghazi K, Chung L (2010) A survey on indoor positioning systems: foreseeing a quality design. In: de Leon F. de Carvalho A, Rodrguez-Gonzlez S, De Paz Santana J, Rodrguez J (eds) Distributed computing and artificial intelligence. Advances in intelligent and soft computing, vol 79. Springer, Berlin, pp 373–380
25. Saltenis S, Jensen CS, Leutenegger ST, Lopez MA (2000) Indexing the positions of continuously moving objects. SIGMOD Rec 29(2):331–342
26. Tao Y, Papadias D (2001) 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, pp 431–440, San Francisco, CA, USA
27. Tao Y, Papadias D, Sun J (2003) The TPR*-tree: an optimized spatio-temporal access method for predictive queries. In: VLDB, pp 790–801. http://www.vldb.org/conf/2003/papers/S24P01.pdf

28. Theodoridis Y, Vazirgiannis M, Sellis TK (1996) Spatio-temporal indexing for large multimedia applications. In: ICMCS, pp 441–448. http://dblp.uni-trier.de
29. Waluyo AB, Srinivasan B, Taniar D (2005) Research in mobile database query optimization and processing. Mobile Inf Syst 1(4):225–252
30. Wolfson O, Xu B, Chamberlain S, Jiang L (1998) Moving objects databases: issues and solutions. In: Proceedings of tenth international conference on scientific and statistical database management, 1998, pp 111–122
31. Yellen J, Gross JL (2005) Graph theory and its applications. Chapman and Hall/CRC, New York, pp 585–655
32. Yuan W, Schneider M (2010) Supporting continuous range queries in indoor space. In: Proceedings of the 2010 eleventh international conference on mobile data management, MDM '10, pp 209–214, Washington, DC, USA. IEEE Computer Society
33. Zhao G, Xuan K, Rahayu W, Taniar D, Safar M, Gavrilova ML, Srinivasan B (2011) Voronoi-based continuous *k* nearest neighbor search in mobile navigation. IEEE Trans Ind Electron 58(6):2247–2257