Indexing the Trajectories of Moving Objects in Symbolic Indoor Space

Christian S. Jensen[1], Hua Lu[1], and Bin Yang[1,2]
[1] Department of Computer Science, Aalborg University, Denmark
[2] School of Computer Science, Fudan University, China
{csj,luhua,yang}@cs.aau.dk

Abstract. Indoor spaces accommodate large populations of individuals. With appropriate indoor positioning, e.g., Bluetooth and RFID, in place, large amounts of trajectory data result that may serve as a foundation for a wide variety of applications, e.g., space planning, way finding, and security. This scenario calls for the indexing of indoor trajectories. Based on an appropriate notion of indoor trajectory and definitions of pertinent types of queries, the paper proposes two R-tree based structures for indexing object trajectories in symbolic indoor space. The RTR-tree represents a trajectory as a set of line segments in a space spanned by positioning readers and time. The TP^{2}R-tree applies a data transformation that yields a representation of trajectories as points with extension along the time dimension. The paper details the structure, node organization strategies, and query processing algorithms for each index. An empirical performance study suggests that the two indexes are effective, efficient, and robust. The study also elicits the circumstances under which our proposals perform the best.

1 Introduction

People spend large parts of their lives in indoor spaces such as office buildings, shopping centers, conference facilities, airports, and other transport infrastructures. At the same time, such spaces are becoming increasingly large and complex. For example, the New York City Subway has 468 stations and a network of 842 miles. Each day, the subway serves more than 6 million users, totalling 2+ billion annually.

With the deployment of indoor positioning based on technologies such as RFID [23], Bluetooth [8], and Wi-Fi [3], large volumes of tracking data are becoming available that enable a range services akin to those enabled by GPS-based positioning in outdoor settings. Example services include indoor navigation, personal security, and those providing insight into how and how much the indoor space is being used, which is important in planning applications and for the pricing of advertisement space and store rentals. Motivated by these observations, this paper provides two techniques for the indexing of the trajectories of objects moving in symbolic indoor space.

Over the past decade, much research has been devoted to outdoor applications involving moving objects [9, 24], and substantial research concerns the indexing and querying of the positions of moving objects [2, 6, 12–14, 18, 20, 21, 25] and their trajectories [5, 17]. However, the outcomes of this body of research is not easily applicable in indoor scenarios. First, indoor space is typically modeled differently from outdoor space, where either Euclidean space or a spatial network is typically assumed. Indoor space is characterized by entities such as doors, rooms, and hallways that enable and constrain movement. This renders movement more constrained than outdoor Euclidean movement. Consequently, geometric representations, e.g., the linear model that is widely adopted for describing outdoor movement, are not suitable for describing

indoor movement. Further, indoor movement is less constrained than outdoor spatial-network movement, where the position of an object is constrained to a position on a polyline. As a result, symbolic models, rather than geometric models, of indoor space are often used [4], which renders indexes for outdoor moving objects inapplicable.

Second, indoor positioning technologies differ fundamentally from those typically assumed in outdoor settings. Unlike GPS and cellular technologies, which are capable of continuously reporting the position and velocity of an object with varying accuracies, we assume indoor positioning technologies that rely on proximity analysis [11] and are able to report neither velocities nor exact locations. In particular, an indoor moving object is detected only when it enters the sensing or activation range of a positioning device, e.g., an RFID reader, or a Bluetooth base station.

We propose two R-tree based structures for indexing the trajectories of objects moving in symbolic indoor space, together with algorithms for the processing of pertinent queries, including spatiotemporal range and topological queries. We assume a symbolic representation of indoor space and use RFID for positioning. The trajectory of an object is then represented as a series of records, each of which indicates that the object is within the activation range of a specific RFID reader during a period of time.

The RTR-tree, similar to the R-tree, uses a specific node organization. It organizes a trajectory as a set of line segments in the plane spanned by positioning readers and time. For each type of query, a corresponding geometrical representation is derived and used to search the RTR-tree similarly to how an R-tree is searched.

The TP^{2}R-tree uses the same underlying space, but applies a data transformation that turns a trajectory into a set of points in the plane, augmented with temporal extents. The objective is to obtain a better node organization. When a node overflows during insertion, splitting is handled by taking the time extents into consideration such that fewer node accesses are expected by subsequent queries. Efficient query processing algorithms for the TP^{2}R-tree are also detailed.

A comprehensive performance study on synthetic and real data sets is conducted to evaluate the two indexes, together with relevant query processing algorithms. A wide range of parameter settings are applied. The results show that tree construction is scalable with respect to the trajectory data size and that query processing is quite efficient.

The paper's contributions may be summarized as follows. The paper formalizes a moving-object trajectory model and trajectory-related queries in symbolic indoor space. It presents two index structures with different node organization strategies that apply specifically to indoor trajectories. It presents accompanying algorithms for the processing of pertinent queries. Finally, it presents the results of a comprehensive and favorable performance study of the paper's proposals.

The rest of this paper is organized as follows. Section 2 describes existing indexes for moving objects. Section 3 presents the assumed data model, the assumed representation of indoor trajectories, and the queries considered. Sections 4 and 5 detail the RTR-tree and and TP^{2}R-tree, respectively. Section 6 reports on the relevant experimental study. Section 7 concludes and offers research directions.

2 Existing Moving-Object Indexes

Most research on outdoor moving-object indexing assumes, explicitly or implicitly, the availability of GPS-type positioning. A GPS receiver can continuously (typically

each second) report location (longitude and latitude) and velocity. Hence, the trajectory of a GPS-equipped outdoor moving object is usually modeled as a polyline in three-dimensional space-time space.

The TB-tree (Trajectory Bundle) [17] extends the R-tree to allow the indexing of such trajectories by allowing a leaf node to contain line segments only from the same trajectory, which facilitates retrieval of the trajectory of an individual object, but adversely affects spatiotemporal range queries. SETI (Scalable and Efficient Trajectory Index) [5] is based on a static partitioning of the spatial dimensions. The aim is that trajectory line segments in the same index partition belong to the same trajectory. Then, within each partition, all line segments are indexed by an R-tree. The performance of SETI depends heavily on the partitioning function used. Because indoor positioning technologies only determine the presence of moving objects at predefined locations, the polyline representation used for outdoor trajectories is inapplicable for indoor trajectories. As a result, the TB-tree and SETI cannot be applied to indoor trajectories.

In addition to indexes directly targeting trajectories, proposals exist for the indexing of historical positions of outdoor moving objects in order to support spatiotemporal range queries. The 3D R-tree [22] treats the time dimension as yet another spatial dimension. The HR-tree (historical R-tree) [16] adds a temporal dimension to the R-tree by means of replication when updates occur, so that multiple R-trees result, each corresponding to a time interval. A tree node belongs to multiple consecutive trees if the data in it remains constant during the time interval associated with the trees. The MV3R-tree (Multi-version 3D R-tree) [20] integrates the HR-tree and the 3D R-tree. The PPR-tree (Partially-Persistent R-tree) [14] indexes both historical and current positions of animated objects by applying multi-versioning.

The idea behind the 3D R-tree, of treating time simply as an additional dimension, can be used to index indoor trajectories. Our RTR-tree with its basic node organization strategies (see Section 4.2) adopts this approach. However, our studies show that this straightforward approach yields poor performance. The HR-tree is unsuitable for indexing indoor trajectories for three reasons. First, a mapping from each indoor positioning device to its sensing range is needed for indoor positioning data to be indexed by an HR-tree. Second, the limited device sensing ranges cause high data volatility, as few objects remain for long within a range. This considerably reduces the overlapping among tree nodes that the HR-tree exploits. Third, the HR-tree prefers time point queries over time interval queries. Our proposals aim to support both query types efficiently.

Other indexes for the current and future positions of outdoor moving objects [2, 6, 12, 13, 18, 25] are unsuitable for indoor trajectory indexing. Moreover, almost all assume a linear movement model, which is unavailable for indoor objects. In particular, although the BB^x-tree [15] and the R^{PPF}-tree [19] index the historical, current, and future positions of moving objects, they both assume a linear movement model.

3 Data Model, Trajectory Representation, and Queries

3.1 Data Model and Queries

As suggested earlier, the geometric polyline representation used for outdoor trajectories is not ideal for indoor trajectories. For example, assume that an object moves from one room to another so that two consecutive location reports are in different rooms. (Due

to the limitations of indoor positioning, locations between these two reports are not obtained.) Using the polyline representation, the line segment between the two reported locations is unlikely to intersect with the door, but will go through the wall between the rooms. This means that the trajectory has the moving object going through the wall, which contradicts reality and renders the trajectory of limited use.

Typical queries on indoor trajectories contain symbolic references such as room numbers rather than simply geometric locations. For example, we may be interested in determining which room a moving object was in at a specific time, or the sequence of rooms a moving object visited during a given time interval. Such considerations lead to an indoor trajectory model that is composed of trajectory records in the format $(objectID, symbolicID, \mathbb{T})$. Here, $objectID$ is the identifier of a moving object; $symbolicID$ is the identifier for a specific indoor space region (e.g., a room); and $\mathbb{T}$ indicates the time, either a time instant or a time interval.

We consider two types of queries. Given an indoor spatial extent E_s and a temporal extent E_t, an *indoor range query* Q(E_s, E_t) returns all trajectory records that intersect the spatiotemporal region defined by E_s and E_t: $Q(E_s, E_t) \rightarrow \{trajectory\ records\}$.

Specifically, E_s can be represented by either a subset of symbolic references (e.g., room numbers) or a subset of Euclidean space (e.g., a polygon or a circle in a floor plan). Next, E_t indicates a temporal extent, which is either a time instant or a time interval. For example, query $Q(room_1$, [1:00 p.m., 1:15 p.m.]) returns all trajectory records indicating that the corresponding objects were in $room_1$ at some time between 1:00 p.m. and 1:15 p.m. A Euclidean spatial constraint in a query can be transformed to one or more symbolic references. This is covered in detail in Section 4.3.

Next, given an indoor space partition (e.g., a room), a temporal extent, and a topological predicate, an *indoor topological query* returns all objects whose trajectories satisfy the given predicate with respect to the space partition within the given temporal extent: $Q(E_s, E_t, P) \rightarrow \{objectID\}$.

Here, E_t is as before, and E_s is an indoor space partition with the property that positioning devices are deployed such that the satisfaction of predicate P can be detected or inferred from the positioning data. (Issues related to deployments are addressed in Section 4.3.) Moreover, P denotes a topological predicate such as (a) *enter*, (b) *leave*, or (c) *cross* [7, 17]. We use the cross predicate to mean enter and then leave, or leave and then enter. For example, query $Q(room_1$, [1:00 p.m., 1:15 p.m.], *enter*) returns all objects that entered $room_1$ between 1:00 p.m. and 1:15 p.m.

The range and topological queries can work as building blocks for constructing more complex queries. Such queries can be processed by combining the algorithms presented in this paper.

3.2 RFID Based Indoor Positioning

Our focus is on positioning by means of RFID technology [23], and we assume a setting where RFID readers are deployed at fixed locations, e.g., at building entrances, doors, hallways, while RFID tags are attached to moving objects. An RFID reader employs proximity analysis [11] to detect an RFID tag when the tag (with the object to which it is attached) enters the reader's activation range. Different readers support different sensing ranges. The position of each reader is recorded in the database after deployment.

Each RFID reader continuously detects and reports tags with a frequency determined by its sampling rate, a hardware specific parameter that usually varies from 1 to 3 times per second [23]. We use T_S to denote the RFID reader sampling period. The raw RFID readings are of the format $(readerID, tagID, t)$, meaning that a reader $readerID$ detects the moving object with tag $tagID$ in its activation range at timestamp t. Our proposals accommodate the possibility of a tag being detected simultaneously by multiple readers.

Unlike GPS that reports accurate geographic locations, RFID-based positioning only reports the presences of objects in readers' activation ranges. An object's location during the time in-between consecutive RFID readings cannot be inferred from the reading sequence without additional information such as the floor plan. Also, different deployments of RFID readers in the same indoor space generally result in different positioning accuracies. Positioning reader deployment is beyond the scope of this paper. Rather, our focus is to enable indexing of whatever RFID positioning data is available.

Given a raw RFID reading sequence, a trajectory table can be constructed that contains records of the following format: $(recordID,\ tagID, readerID, t_s, t_e)$. Here, $recordID$ is the identifier of each trajectory record, and t_s (t_e) indicates the first (last) time point when reader $readerID$ detects tag $tagID$ in its activation range. This representation removes the "duplicate" readings caused by the sampling. A table containing trajectories is shown in Table 1.

Table 1. Indoor Trajectories

recordID	tagID	readerID	t_s	t_e
rd_1	tag_1	$reader_1$	t_1	t_3
rd_2	tag_3	$reader_2$	t_2	t_4
rd_3	tag_2	$reader_3$	t_3	t_5
rd_4	tag_3	$reader_2$	t_6	t_8
rd_5	tag_2	$reader_2$	t_7	t_9
rd_6	tag_1	$reader_4$	t_{10}	t_{11}
rd_7	tag_3	$reader_3$	t_{11}	t_{12}

We proceed to propose two indexes for indoor trajectories. In Section 4 we detail the RTR-tree which, treating trajectories as sets of line segments, extends the R-tree with specific node organization strategies. In Section 5, we detail the TP^{2}R-tree that is intended to improve the tree organization by representing trajectories as points with time extension parameters.

4 RTR-Tree: Reader-Time R-Tree

The *Reader-Time R-tree* (*RTR-tree*) that is essentially a two dimensional R-tree [10] on the Reader-Time space. The vertical axis, which we call the *reader axis*, represents reader identifiers. The horizontal axis, which we call the *time axis*, represents time. Using this space, an indoor trajectory record becomes as a horizontal line segment.

4.1 RTR-Tree Index Structure

Leaf nodes in the RTR-tree contain index entries of the form $(MBR,\ recordID)$, where MBR is the minimum bounding rectangle of the record, identified by $recordID$. An MBR in a leaf node is a line segment of the form $MBR(readerID,\ t_s,\ t_e)$, where $readerID$ indicates a reader on the Reader axis and t_s and t_e (coming from the corresponding trajectory record) indicate the (closed) time interval $[t_s,\ t_e]$. A leaf entry implies that a specific tag is detected by $readerID$ between timestamps t_s and t_e.

Non-leaf nodes of the RTR-tree have entries of the form $(MBR,\ cp)$, where cp is a pointer to a child node in the RTR-tree and MBR is the minimum bounding rectangle that contains all MBRs in the child node's entries. Each MBR in a non-leaf node is in the form $MBR(readerID_{min},\ readerID_{max},\ t^{\vdash},\ t^{\dashv})$, where $readerID_{min}$ and

$readerID_{max}$ indicate the interval [$readerID_{min}$, $readerID_{max}$] on the Reader axis and $t^{\vdash}$, $t^{\dashv}$ indicates the time interval [$t^{\vdash}$, $t^{\dashv}$]. Therefore, a non-leaf entry implies that between times $t^{\vdash}$ and $t^{\dashv}$, some tags are detected by some readers whose identifiers fall inclusively into the range [$readerID_{min}$, $readerID_{max}$].

Since the spatial information is represented by readers, it is beneficial to order the reader identifiers on the Reader axis according to their spatial proximity. Space-filling curves, e.g., the Hilbert curve, are often used to map objects in a higher dimensional space to one-dimensional space in order to preserve their proximity as best as possible [6, 12]. However, space-filling curves are less attractive for use in indoor space because the obstacles prevalent in indoor settings and the constrained topology of indoor space tend to result in locations being close in a Euclidean sense not actually being close. For example, two close locations on either side of a wall may be relatively far apart when taking into account the topology.

Instead, we propose to take into account the indoor topology when assigning identifiers to readers. Our objective is to assign consecutive identifiers to readers in the same partition, to readers in adjacent partitions, and to readers on the same floor. Given a specific indoor space and reader deployment, a variety of different orderings can be envisioned that target this objective.

4.2 Node Organization Strategies

Insertion of new index entries into an RTR-tree is carried out as in the R-tree: new index entries are added into leaf nodes, nodes that overflow are split, and splits may propagate up the tree. The R-tree's strategies for identifying an appropriate leaf node and for node splitting are applied here, using the area of each MBR to guide node organization.

We calculate the area of an MBR in the RTR-tree in two different ways: (1) $Area = (readerID_{max} - readerID_{min}) * (t^{\dashv} - t^{\vdash})$; (2) $Area^{+} = (readerID_{max} - readerID_{min} + 1) * ((t^{\dashv} - t^{\vdash})/T_S + 1)$. The $Area$ formula computes the geometric area and is identical to the method used by the original R-tree. In this formula, the area of any MBR of records with the same reader is 0 as the time dimension is neglected. Consequently, records with the same reader are put into the same node (resulting from leaf node choosing and node splitting), even if they are considerably apart in the time dimension. We call this the basic strategy.

The $Area^{+}$ formula takes into account the number of possible raw readings. Each point in the Reader-Time coordinate system indicates a possible raw reading. By adding 1 (and not any other arbitrary positive number), the $Area^{+}$ formula actually calculates the number of possible raw readings in an MBR. This calculation renders each node to have as few raw readings as possible. This way, we avoid too many zero-area MBRs that disable the use of area differences for leaf node selection and node splitting. We call this strategy *RTR-tree Area*$^{+}$.

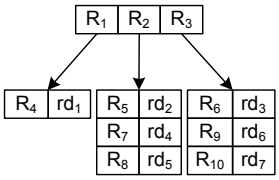


Fig. 1. RTR-Tree, Basic

For example, let the capacity of each node be 3. After all the trajectory records shown in Table 1 are inserted into the RTR-tree using the basic strategy, the the RTR-tree and MBRs are shown in Figures 1 and 2, respectively. Using the Area^{+} strategy results in the MBRs shown in Figure 3. Note that each r_i on the Reader axis corresponds to a reader identifier $reader_i$.

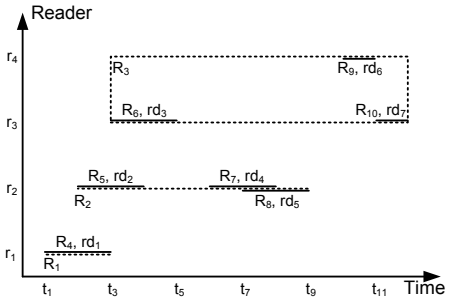


Fig. 2. MBRs of the RTR-Tree, Basic

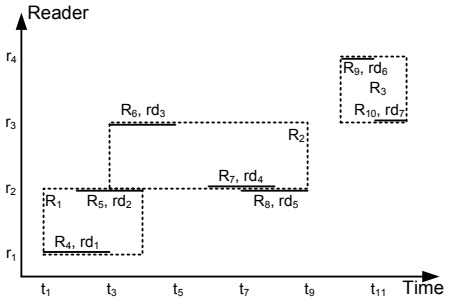


Fig. 3. MBRs of the RTR-Tree, Area$^+$

4.3 Query Processing

Given a query as described in Section 3.1, it is first transformed according to its type such that a corresponding geometric representation g is obtained that describes the query in the Reader-Time coordinate system and is used to process the query. Particularly, a recursive depth-first tree search is used on the RTR-tree to process the query. The search involves all nodes whose MBR overlaps with g. Due to space limitations, we omit the details of the search algorithm. We proceed to detail how to obtain the geometric representations of all query types.

Range Queries Transformation Given a range query $Q(E_s, E_t)$, two possibilities exist regarding the representation of the spatial extent E_s.

Symbolic Representation. If the spatial extent E_s is represented in symbolic space, i.e., by $readerID$s, no matter which form the temporal extent E_t takes, the query can be transformed into basic geometric shapes, which is described in Table 2. *Single ReaderID* indicates that E_s is represented as a single $readerID$; *Continuous ReaderIDs* indicates that E_s is represented as a set of $readerID$s that are continuous on the vertical Reader axis. A query with a single reader identifier (i.e., query types QT_1 and QT_2) and continuous reader identifiers (i.e., query types QT_3 and QT_4) can be executed directly on the RTR-tree. Example representations for queries with different combination of E_s and E_t are shown in Table 2 and Figure 4.

Table 2. Symbolic Range Query Transformation for RTR-Tree

Time		**Single ReaderID**	**Continuous ReaderIDs**
Instant	**Query Format**	*Query Type 1:* $QT_1(readerID, t)$	*Query Type 3:* $QT_3([readerID_m, readerID_n],\ t)$
	Geometry Representation	Point	Vertical line segment
	Example	$Q_1(reader_2,\ t_3)$	$Q_3([reader_1, reader_3],\ t_6)$
Interval	**Query Format**	*Query Type 2:* $QT_2(readerID, [t_i, t_j])$	*Query Type 4:* $QT_4([readerID_m, readerID_n], [t_i, t_j])$
	Geometry Representation	Horizontal line segment	Rectangle
	Example	$Q_2(reader_3, [t_2, t_4])$	$Q_4([reader_1, reader_3],\ [t_7, t_9])$

Euclidean Representation. If the spatial extent E_s is represented in Euclidean space, it must be mapped to symbolic space. For example, the spatial extent of query Q_5 is represented as a dashed rectangle in Figure 5; the circle surrounding each reader identifier indicates the reader's activation range. If a reader's activation range is covered by E_s then the reader-observed tags are definitely in E_s, e.g., $reader_2$ and $reader_3$.

Next, if a reader's activation range overlaps with E_s then the reader-observed tags are possibly in E_s, e.g., $reader_1$ and $reader_4$. Finally, if a reader's activation range is disjoint from E_s then the reader-observed tags are definitely not in E_s, e.g., $reader_5$.

Thus, we transform E_s to two separate sets of reader identifiers: S_p contains all readers partially overlapping E_s, and S_d contains all readers fully covered by E_s. In Figure 5, $S_p = \{reader_1, reader_4\}$; $S_d = \{reader_2, reader_3\}$. If the temporal extent of Q_5 is a time interval, Q_5 will be transformed into a type 4 query $Q'_5([reader_1, reader_4], E_t)$. After Q'_5 is processed, a refinement can be applied to distinguish results obtained from different reader sets.

The example implicitly assumes that the reader identifiers are ordered according to their spatial proximity as discussed in Section 4.1, which results in a single type 4 query. Generally, an Euclidean range query like Q_5 can be transformed into several queries of type 2 and/or type 4 depending on the reader identifiers involved. More transformed queries are expected to incur higher processing costs. We investigate the effect of reader identifer ordering on Euclidean range queries in Section 6.3.

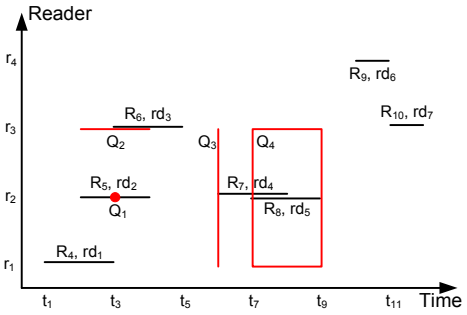


Fig. 4. Symbolic Range Queries in the RTR-Tree

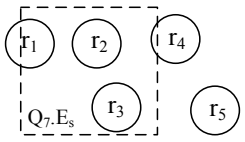


Fig. 5. Euclidean Range Queries in the RTR-Tree

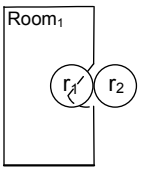


Fig. 6. Reader Deployment

Indoor Topological Queries Transformation Similarly to how Euclidean range queries are handled, indoor topological queries can be transformed to symbolic range queries. However, the transformations are usually more complex than for Euclidean range queries because knowledge of the specific RFID reader deployment is needed in order to do such transformations for indoor topological queries.

To be specific, a pair of readers are required to be deployed on a door to detect the movement direction of an object. Based on this, entering or leaving a room can be determined from the RFID readings. A possible deployment is shown in Figure 6. If an object enters $room_1$, it will be observed by $reader_2$ and then $reader_1$; if the object leaves $room_1$, it will be observed first by $reader_1$.

With this type of reader deployment, we are able to process indoor topological queries. An indoor topological query $Q(E_s, E_t, P)$ can be transformed and processed by the procedure described in Algorithm 1. Here, the given topological query is transformed into two type 2 range queries (lines 2–3) on two readers. If an object appears in the results of both queries and the corresponding time intervals overlap, it is added to the result of the given topological query (lines 4–8).

For different types of topological queries, as indicated by the predicate P, we need to input the two readers in the correct order when calling EnterLeaveQuery. We let the

reader inside the room (door) be rin and let the one outside be $rout$. If P is *enter*, $rout$ should be used as $reader_{first}$ followed by rin as $reader_{second}$. If P is *leave*, the two readers should be switched.

Algorithm 1 EnterLeaveQuery(ReaderID $reader_{first}$, ReaderID $reader_{second}$, Timestamp t_i, Timestamp t_j)

```
1: TagIDSet result;  RecordSet R1 ← ∅;  RecordSet R2 ← ∅;
2: Execute query Q2(reader_first, [t_i, t_j]), get result into R1;
3: Execute query Q2(reader_second, [t_i+T_S, t_j]), get result into R2;
4: for each record r_i ∈ R1 do
5:    for each record r_j ∈ R2 do
6:       if r_i.tagID = r_j.tagID and r_j.t_s ≤ r_i.t_e + T_S ≤ r_j.t_e then
7:          Add tagID to result
8: return result;
```

Refer to the example shown in Figure 6. An indoor topological query Q($room_1$, [t_i, t_j], *enter*), which is intended to find those objects that enter *room*$_1$ during time period [t_i, t_j], can be executed as EnterLeaveQuery($reader_2$, $reader_1$, t_i, t_j). Similarly, a leave query Q($room_1$, [t_i, t_j], *leave*) can be executed as EnterLeaveQuery($reader_1$, $reader_2$, t_i, t_j). A cross query Q($room_1$, [t_i, t_j], *cross*) can be executed as an intersection of two queries: EnterLeaveQuery($reader_1$, $reader_2$, t_i, t_j) $\cap$ EnterLeaveQuery ($reader_2$, $reader_1$, t_i, t_j), which returns those objects that entered and then left the room within the given time period.

5 TP^{2}R-Tree: Time Parameter Point R-Tree

An indoor trajectory record is a horizontal line segment in Reader-Time space. The efficiency of processing a query in an R-tree based index depends on the areas of the MBRs in the index. It is thus generally beneficial to minimize the MBRs. To achieve this, we represent the horizontal line segments as more compact points with a time parameter indicating their lengths along the time axis. Then we index the resulting points using an R-tree that is modified to take the time parameters into account.

5.1 TP^{2}R-Tree Index Structure

A leaf node in the TP^{2}R-tree contains index entries of the form (MBR, Δt, $recordID$), where MBR indicates the minimum bounding rectangle of the corresponding record identified by $recordID$, and Δt is a time parameter that indicates the duration of the continuous reading by the same reader. For each trajectory record, Δt thus equals $t_e - t_s$. Since each record is represented as a point, a leaf-node MBR in is a point of the form $MBR(readerID, t_s)$, where $readerID$ is a reader on the Reader axis and t_s is a time point on time axis.

The non-leaf nodes contain index entries of the form (MBR, Δt, cp), where MBR covers all rectangles in the child node's entries, formatted as in the RTR-tree; cp is a child pointer; and Δt is a time parameter. If cp points to a leaf node N_l, Δt is represented as follows:

$$\max_{\forall e_i \in N_l}(e_i.MBR.t_s + e_i.\Delta t) - \max_{\forall e_j \in N_l}(e_j.MBR.t_s)$$

If cp points to a non-leaf node N_n, Δt is represented as follows:

$$\max_{\forall e_i \in N_n} (e_i.MBR.t^{\dashv} + e_i.\Delta t) - \max_{\forall e_j \in N_n} (e_j.MBR.t^{\dashv})$$

This way, Δt indicates the tightest bound that covers all subnodes on the time axis.

5.2 Node Organization Strategies

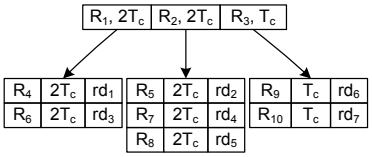


Fig. 7. TP^{2}R-Tree, Basic

Insertion is done using the same framework as for the RTR-tree. We consider three node organization strategies. The first two are based on the least area enlargement. The area calculation of the first strategy is based on the $Area$ formula; we call this the basic strategy.

Given a node capacity of 3, the tree and MBRs for the basic strategy are exemplified in Figures 7 and 8. For simplicity, T_c denotes $t_{i+1} - t_i$ for any two consecutive timestamps.

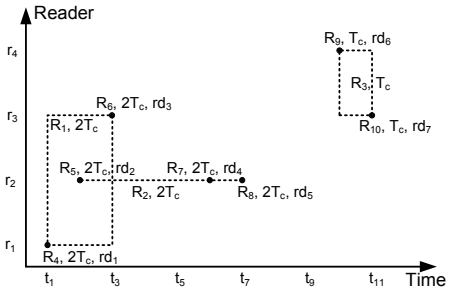


Fig. 8. MBRs of the TP^{2}R-Tree, Basic

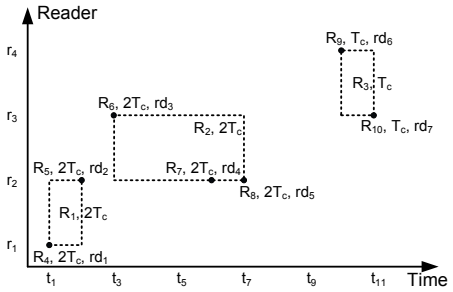


Fig. 9. MBRs of the TP^{2}R-Tree, Area$^+$

The second strategy, ***TP^{2}R-tree Area$^+$***, calculates areas using the $Area^+$ formula. Figure 9 illustrates the MBRs resulting from the TP^{2}R-tree $Area^+$ strategy.

The third strategy, ***TP^{2}R-tree Split2***, uses specific **ChooseLeaf** and **SplitNode** algorithms. We first need to define the virtual minimum bounding rectangle (VMBR) of an entry e in non-leaf node of the TP^{2}R-tree: $VMBR(readerID_{min}, readerID_{max}, t^{\vdash}, t^{\dashv})$. Here, $readerID_{min}$ and $readerID_{max}$ are the same as the MBR of the entry, $VMBR.t^{\vdash}$ equals $MBR.t^{\vdash}$, and $VMBR.t^{\dashv}$ equals $MBR.t^{\dashv} + e.\Delta t$. The VMBR of MBR R_1 in Figure 8 is shown in Figure 10.

Referring to the analysis of the query types in Section 4.3, the basic queries are query type 1 and query type 2. These queries involve merely one specific reader. Thus, we try to place entries with same or near-same readerID in the same node. The TP^{2}R-tree Split2 strategy chooses the leaf node whose MBR needs the least Reader dimension enlargement to include the new index entry, and it resolves ties by choosing the entry whose VMBR areas need the least area enlargement (using *Area$^+$* formula) to include the new index entry. Due to space limitations we omit the ChooseLeaf Algorithm.

The basic criterion for node splitting is the same as for the R-tree, minimizing the probability that both new nodes will be examined in subsequent queries. The pseudo code of **SplitNode** is shown in Algorithm 2. It discriminates on the Reader dimension

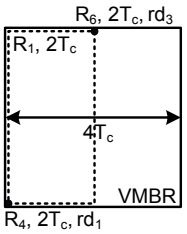


Fig. 10. VMBR, TP^{2}R-Tree

first, selects two entries containing the biggest (smallest) readerID, and assigns them to two groups as seeds (lines 2–3). The assignment of the remaining entries gives priority to satisfying the requirement of having a minimum number of objects in each group (lines 5–6). Next, if there is an entry with a readerID range in that of one of the groups, the entry is assigned to that group (lines 9–12). If such an entry is chosen, a new iteration of the while loop will be invoked, ensuring no groups with underflow.

If no such entries exist, both Reader and time dimensions are considered using the $Area^+$ definition. For each remaining entry, the VMBR $Area^+$ increment of either group needed to include the entry is calculated (lines 15–16). From all remaining entries, we choose the one with the maximum difference of two VMBR $Area^+$ increments, and we assign it to the group that needs least VMBR Area$^+$ enlargement (lines 17–18). The resulting MBRs of the TP^{2}R-tree example are shown in Figure 11.

Algorithm 2 SplitNode (EntrySet $SetE$)

1: EntrySet $GroupB \leftarrow \emptyset$, $GroupS \leftarrow \emptyset$;
2: Move the entry e_b with the biggest $readerID$ from $SetE$ into $GroupB$;
3: Move the entry e_s with the smallest $readerID$ from $SetE$ into $GroupS$;
4: **while** $SetE \neq \emptyset$ **do**
5: **if** One group has so few entries that all the rest must be assigned to it in order for it to have the minimum number **then**
6: Move all entries from $SetE$ to that group; **break**;
7: Boolean $flag \leftarrow$ FALSE;
8: **for** each entry e in $SetE$ **do**
9: **if** The reader range of $e.MBR$ is in the reader range of $GroupB.MBR$ **then**
10: Move e from $SetE$ into $GroupB$; $flag \leftarrow$ TRUE; **break**;
11: **else if** The reader range of $e.MBR$ is in the reader range of $GroupS.MBR$ **then**
12: Move e from $SetE$ into $GroupS$; $flag \leftarrow$ TRUE; **break**;
13: **if** $\neg flag$ **then**
14: **for** each entry e in $SetE$ **do**
15: Calculate d_1, the VMBR Area$^+$ increment of $GroupB$ to include the entry $e.MBR$
16: Calculate d_2, the VMBR Area$^+$ increment of $GroupS$ to include the entry $e.MBR$
17: Choose the entry e_m with the maximum $|d_1 - d_2|$.
18: Move e_m from $SetE$ to the group whose VMBR Area$^+$ needs the least enlargement.

Similarly to the query transformation used in the RTR-tree, queries here can also be transformed into the Reader-Time coordinate representation. However, the TP^{2}R-tree differs from the RTR-tree in that its entries have both an MBR and a VMBR. A query not overlapping with an entry's MBR may overlap with the entry's VMBR, qualifying some records in that entry for the query. We thus need to modify the RTR-tree's search algorithm to accommodate the time parameter and the VMBRs in the TP^{2}R-tree.

5.3 Expansion-Based Query Processing

The idea is to expand the query geometry in the temporal dimension while still using depth-first search. In particular, when checking whether a TP^{2}R-tree entry e overlaps with the given query geometry g, we use an expanded version of g. Table 3 shows in

Table 3. Query Geometry Expansion for TP^{2}R-Tree

Original Query	**Expanded Query**
Type 1: $QT_1(readerID, t)$ E.g., $Q_1(reader_3, t_4)$	*Type 2:* $QT_2(readerID, [t - e.\Delta t, t])$ E.g., Q_1'$(reader_3, [t_2, t_4])$
Type 2: $QT_2(readerID, [t_i, t_j])$ E.g., $Q_2(reader_2, [t_9, t_{10}])$	*Type 2:* $QT_2(readerID, [t_i - e.\Delta t, t_j])$ E.g., Q_2'$(reader_2, [t_7, t_{10}])$
Type 3: $QT_3([readerID_m, readerID_n], t)$ E.g., $Q_3([reader_1, reader_2], t_4)$	*Type 4:* $QT_4([readerID_m, readerID_n], [t - e.\Delta t, t])$ E.g., Q_3'$([reader_1, reader_2], [t_2, t_4])$
Type 4: $QT_4([readerID_m, readerID_n], [t_i, t_j])$ E.g., $Q_4([reader_3, reader_4], [t_{12}, t_{13}])$	*Type 4:* $QT_4([readerID_m, readerID_n], [t_i - e.\Delta t, t_j])$ E.g., Q_4'$([reader_3, reader_4], [t_{11}, t_{13}])$

detail how a query geometry g is expanded with respect to an encountered entry e. For each query type, the query geometry representation in the Reader-Time coordinate system is expanded to the left horizontally by the Δt value in a given entry. The examples in the table are also illustrated in Figure 12, with the top-level MBRs from the TP^{2}R-tree with the basic strategy (shown in Figure 8).

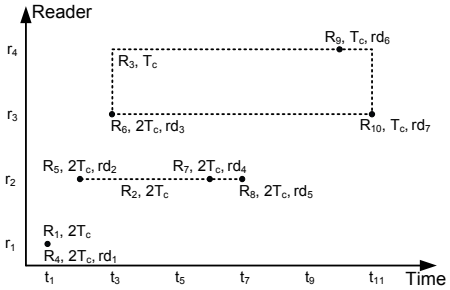


Fig. 11. MBRs of the TP^{2}R-Tree, Split2

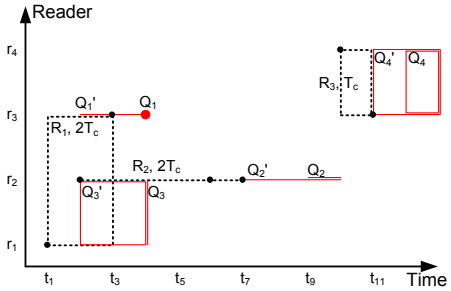


Fig. 12. Query Expansion for TP^{2}R-Tree

6 Experimental Study

6.1 Experimental Settings

Both indexes, together with all the node organization strategies presented, and the query processing algorithms are implemented in Java. The index implementations are based on the Spatial Index Library [1]. A computer with Windows XP professional, a 2.66GHz Core2 Duo CPU, and 3.25GB main memory is used to run all experiments.

We set the the page size (i.e., the tree node size) to 4096 bytes. This yields 204 (170) entries per non-leaf node and 256 (256) entries per leaf node in the RTR-tree (the TP^{2}R-tree). We investigate both tree construction costs and query processing costs. For the former, the total running time is measured; for the latter, the total number of tree node accesses are measured, as this is proportional to the dominant cost in query processing.

We generate moving objects using a 3-floor building plan with 30 rooms and 3 staircases on each floor. All rooms and staircases are connected by doors to a hallway in a star-like way. An RFID reader is deployed by the door of each room. In addition, readers are also deployed along the hallway and in the staircases. The reader identifiers are assigned as follows. First, multiple readers within a partition (e.g., a room or a hallway) are assigned consecutive identifiers whose ordering represent their physical proximity. Second, on each side of a hallway, adjacent partitions (e.g., rooms) are assigned consecutive identifiers and/or identifier ranges. Third, adjacent floors are assigned consecutive

identifier ranges. All objects move according to two rules: 1) an object in a room or a staircase can go to the hallway through a door, or move inside the same room or staircase; 2) an object in the hallway can move in the hallway, move to a staircase, or move into a room through a door.

Table 4. Data Parameter Settings

Parameters	Settings
Object number	1K, 5K, **10K**, 20K, ..., 50K
Minimum object lifespan	50, **100**, 150, 200, 250 (sec)
Reader activation range	**100**, 150, 200, 250 (cm)

Three data-related parameters are varied. Table 4 lists the settings of these parameters, with default values given in bold. With the default minimum object lifespan and reader activation range, the number of moving objects varies from 1,000 to 50,000, resulting in indoor trajectory tables consisting of between 25K and 1,973K records. When the minimum object lifespan varies between 50 and 250 seconds and with the other parameters set default, the trajectory tables consist of between 253K and 973K records. Accordingly, the simulation period for all experiments varies from 10,650 to 12,586 seconds. In addition, the variation of reader activation range between 100 and 250 centimeters results in trajectory tables consisting of between 292K and 322K records.

6.2 Tree Construction

Tree construction costs are reported in Figure 13. The first two strategies of the TP^{2}R-tree yield the longest running time for tree construction, as they both involve large numbers of area calculations and ΔT calculations. However, the hybrid strategy, TP^{2}R-tree Split2, incurs the least cost for almost all settings. This indicates that the Split2 strategy is effective at simplifying the node splitting during TP^{2}R-tree construction.

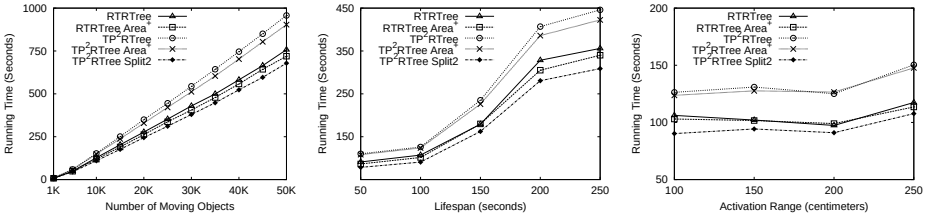
As the number of moving objects increases, the trajectory data size increases accordingly. Therefore, the tree construction cost also increases, as reported in Figure 13(a). Referring to Figure 13(b), a longer object lifespan causes a higher tree construction cost, because a longer lifespan also causes the trajectory data size to increase.

As the reader activation range increases, two different effects occur. On one hand, the time that an object is within a reader's range increases. This may decrease the trajectory data size in the trajectory table, although the numbers of raw RFID readings increases. On the other hand, readers with larger activation ranges tend to detect more objects, thus producing more raw readings and probably more trajectory records. Consequently, the tree construction cost remains steady or decreases slightly as the activation range varies from 100 cm to 200 cm, while the cost increases as the activation range reaches 250 cm, as reported in Figure 13(c).

6.3 Query Processing

Next, we study the query processing costs, and we consider the effects of the ordering of reader identifiers along the Reader dimension.

Results on Range Queries In each batch of experiments in this section, we generate 100 random queries. Unless explicitly stated otherwise, each query is issued with random query parameters. In particular, the time in a type 1 or type 3 query is a random value between 0 and the largest timestamp in the involved set of trajectories. The readerID in a type 1 or type 2 query is chosen randomly among all readerID values. The time interval in a type 2 or type 4 query is a random sub-interval between 0 and the



(a) Effect of Object Number (b) Effect of Object Lifespan (c) Effect of Activation Range

Fig. 13. Tree Construction Costs

largest timestamp in the data set. The readerID range in a type 3 or type 4 query is a random subrange within the range of readerIDs.

Effect of Object Number. We first fix the minimum object lifespan to 100 s and the RFID reader activation range to 100 cm; we then vary the object number from 1,000 to 50,000 to see the effect on the query performance.

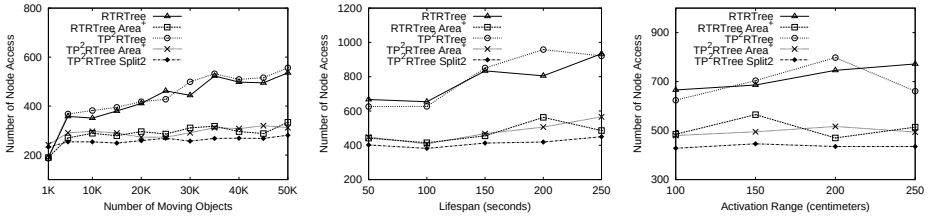
For query type 1 and type 2, the TP^{2}R-tree using the Split2 strategy outperforms all the other trees, as reported in Figure 14(a) and Figure 15(a). The Split2 strategy separates entries with different readerID (ranges) into different nodes after splitting; therefore, the node overlap along the Reader axis is reduced. As a result, the point selections on the Reader axis in query type 1 and type 2 involve fewer tree nodes.

For query type 3 and type 4, the selection on the Reader axis is a range selection, and hence the Split2 strategy loses its clear advantage. As reported in Figure 16(a) and Figure 17(a), both Area$^+$ strategies are always the best (or among the best) as they minimize the number of raw RFID readings in each MBR, which benefits range selections.

Effect of Minimum Object Lifespan. We also vary the minimum object lifespan from 50 to 250 s; the results are reported in Figures 14(b) to 17(b). As a whole, increasing the object lifespan increases the query processing cost for each query type. This is attributed to the increased trajectory data size and the corresponding index sizes that result from the longer object lifespans. Note that the TP^{2}R-tree constructed using the Split2 strategy still outperforms the others for query types 1 and 2, as reported in Figures 14(b) and 15(b). While trees constructed using Area$^+$ strategy prefer query types 3 and 4, as reported in Figures 16(b) and 17(b).

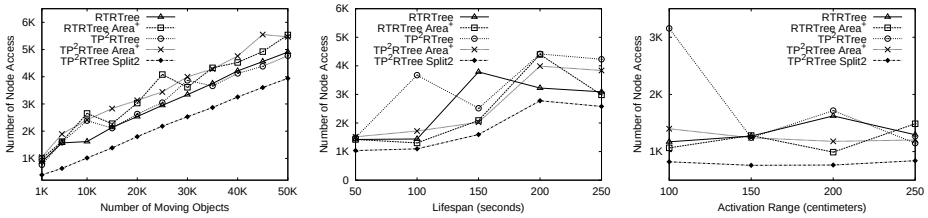
Effect of Reader Activation Range. We then vary the RFID reader activation range from 100 to 250 cm. Referring to Figure 15(c) and Figure 17(c), varied reader activation ranges cause the query processing cost to fluctuate for query type 2 and type 4. This is attributed to two reasons. First, larger reader activation ranges make objects stay longer within a reader's range, thus increasing the time length of each trajectory record and the chance of a record to be in the query result. Second, the range selection on the time axis in query types 2 and 4 then tends to find the answer in fewer nodes as each node becomes "wider" along the time axis.

For query types 1 and 3, as reported in Figures 14(c) and 16(c), the results are relatively more steady as the queries use a point selection on the time axis. For all query types, the Split2 and Area$^+$ strategies outperform the basic ones in almost all settings.



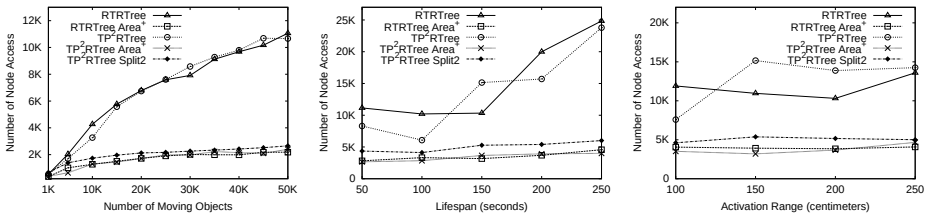
(a) Effect of Object Number (b) Effect of Object Lifespan (c) Effect of Activation Range

Fig. 14. Performance of Query Type 1



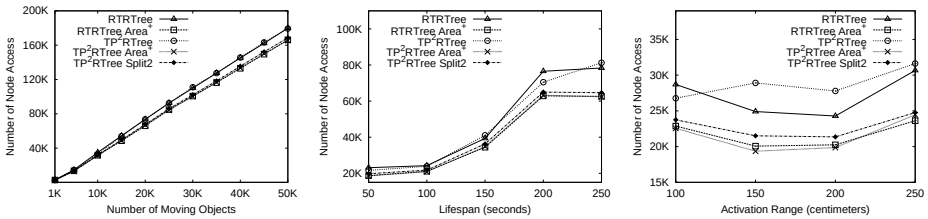
(a) Effect of Object Number (b) Effect of Object Lifespan (c) Effect of Activation Range

Fig. 15. Performance of Query Type 2



(a) Effect of Object Number (b) Effect of Object Lifespan (c) Effect of Activation Range

Fig. 16. Performance of Query Type 3



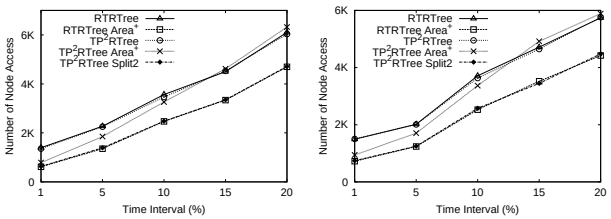
(a) Effect of Object Number (b) Effect of Object Lifespan (c) Effect of Activation Range

Fig. 17. Performance of Query Type 4

Effect of Query Parameters. As the final experiment with range queries, we use data collected in Copenhagen Airport. Although we have assumed RFID technology, this data is obtained using Bluetooth technology (for which the paper's proposals are also applicable). The data set contains more than 500,000 tracking records from 25 Bluetooth hotspots per day. We extract the tracking data on the most active day from April 2008 to October 2008. We vary the query parameters for query types 2, 3 and 4.

For both query types 2 and 4, we vary the time interval from 1% to 20% of the total time span of the set of trajectories. Each query time interval starts at a random time point and is fully within the total time span. According to the results reported in

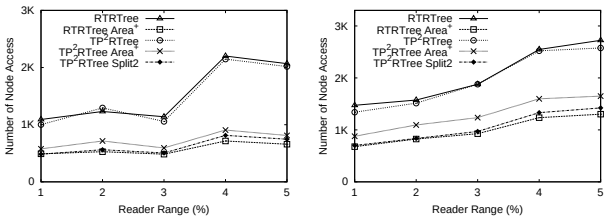
Figure 18(a) and (b), the larger time interval range causes more node access. This is because the large time interval results in a larger query geometry: longer horizontal line segments for type 2 and larger rectangles for type 4. We see that the RTR-Tree Area$^+$ and and TP^{2}R-tree Split2 strategies perform better and degrade more slowly than do the other strategies.



(a) Query Type 2 (b) Query Type 4

Fig. 18. Query Performance vs. Time Interval

We also vary the readerID range for query types 3 and 4, with a random length of 1% to 5% of the difference between the maximum and minimum readerIDs. The results are reported in Figures 19(a) and (b). Larger readerID ranges involve more readers and therefore cause more node accesses. However, the cost increase is moderate for the trees using the Area$^+$ and Split2 strategies.



(a) Query Type 3 (b) Query Type 4

Fig. 19. Query Performance vs. Reader Range

As a whole, the RTR-trees constructed with the Area$^+$ strategy and the TP^{2}R-trees with the Area$^+$ and Split2 strategies perform steadily compared to their alternatives. When using them, the query processing does not degrade markedly as query parameters scale up. This indicates that these node organization strategies result in index trees that are robust to changing query work loads.

Summary. Three important observations follow from the experiments with symbolic range queries. First, compared to the basic node organization strategies, our Area$^+$ and Split2 strategies result in more efficient and robust indexes. Second, the Area$^+$ strategy performs the best for query types 3 and 4, while the Split2 strategy performs the best for query types 1 and 2. Third, the TP^{2}R-tree with Split2 strategy is a quite balanced choice for efficient query processing for all query types.

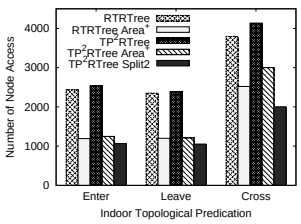


Fig. 20. Effects of Predicates

Topological Queries We also investigate different predicates in topological queries. We pick 35 doors on the first floor as those that are deployed with paired readers. We use a data set with 10,000 objects, 100 cm RFID reader activation ranges, and 100 s minimum object lifespans. For each predicate *enter*, *leave*, and *cross*, we generate 3 queries against each room. In each query, the time interval starts at a random time point, and the interval is set to the lifespan of each object. The resulting node accesses in query processing are reported in Figure 20. The

TP^{2}R-tree Split2 strategy outperforms all others because topological queries are transformed to type 2 range queries favored by the Split2 strategy. The savings are especially apparent for *cross*, as it involves more range queries after the transformation.

Effects of Reader Identifier Ordering We finally consider the effects of varying the reader identifier ordering used in the tree proposals. We generate two data sets using the default settings: 10,000 moving objects, a 100 cm RFID reader activation range, and a 100 s minimum object lifespan. In one set, all RFID reader identifiers are ordered as described in Section 4.1. In the other set, the RFID readers are ordered randomly.

We then generate 10 random Euclidean range queries. The spatial extent E_s in each query is a random rectangle within the floor plan, with an area between 5% and 20% of the floor area. The temporal extent E_t in each query starts at a random time point and lasts 100 s. The results are reported in Figure 21.

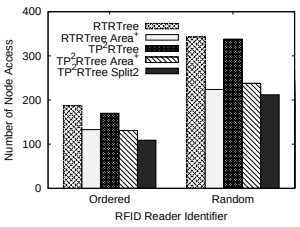


Fig. 21. Effects of ReaderID Ordering on Interval Queries

It is seen that a careful ordering of the RFID reader identifiers improves the query performance considerably. Remember that each Euclidean range query is transformed into a symbolic range query as described in Section 4.3. After the transformation, each set of consecutive reader identifiers will involve only one type 4 query, while any other individual identifier will involve a separate query of type 2. If all reader identifiers are ordered according to spatial proximity, more identifiers are likely to be included in type 4 queries. This results in fewer symbolic queries being executed. For random reader identifiers, the number of symbolic queries after the transformation tends to be large, which yields higher query costs.

7 Conclusion and Future Work

Because of the uniqueness of indoor space and indoor positioning technologies, indoor moving object trajectories are represented differently than traditional outdoor trajectories. Efficient queries against such indoor trajectories require novel indexes. The paper proposes two R-tree based indexes for trajectories of moving objects in symbolic indoor spaces, supporting both spatiotemporal range queries and topological queries. Range queries are processed directly using the indexes; while topological queries are efficiently transformed into range queries. A comprehensive experimental study on both synthetic and real data sets discloses the following findings. First, the two trees are effective indexes for moving object trajectories in symbolic indoor space. Their specific node strategies make them efficient and robust for a wide range of settings. Second, our transformation from topological queries to range queries is effective and efficient. Third, the spatial proximity based ordering of reader identifiers in our trees improves query performance.

Several relevant research directions exist. First, it is possible to extend the TP^{2}R-tree to accommodate on-line trajectories and the current locations of indoor moving objects, as the time extensions in the TP^{2}R-tree can be used to maintain on-line information. Second, it is of interest to apply data mining techniques to indoor trajectories in order to find useful movement patterns for different purposes. The indexes proposed in this paper can be used to facilitate the mining. Third, it is also of interest to adapt the paper's

proposals to other indoor positioning technologies like Wi-Fi, and it is of interest to employ multiple positioning technologies in the same indoor space, such that queries can return more accurate answers.

Acknowledgments This research was partially supported by the Indoor Spatial Awareness project of the Korean Land Spatialization Group and BK21 program. C. S. Jensen is currently a Visiting Scientist at Google Inc.

References

1. Spatial Index Library. http://research.att.com/∼marioh/spatialindex/.
2. P. K. Agarwal, L. Arge, and J. Erickson. Indexing Moving Points. In *Proc. PODS*, pages 175–186, 2000.
3. P. Bahl and V. N. Padmanabhan. RADAR: An In-Building RF-Based User Location and Tracking System. In *Proc. INFOCOM*, pages 775–784, 2000.
4. C. Becker and F. Dürr. On Location Models for Ubiquitous Computing. *Personal Ubiquitous Computing*, 9(1):20–31, 2005.
5. V. P. Chakka, A. Everspaugh, and J. M. Patel. Indexing Large Trajectory Data Sets With SETI. In *Proc. CIDR*, 2003.
6. S. Chen, B. C. Ooi, K.-L. Tan, and M. A. Nascimento. ST^{2}B-Tree: A Self-Tunable Spatio-Temporal B$^+$-Tree Index for Moving Objects. In *Proc. SIGMOD*, pages 29–42, 2008.
7. M. Erwig, and M. Schneider. Developments in Spatio-Temporal Query Languages. In *Proc. DEXA Workshop STDML*, pages 441–449, 1999.
8. S. Feldmann, K. Kyamakya, A. Zapater, and Z. Lue. An Indoor Bluetooth-Based Positioning System: Concept, Implementation and Experimental Evaluation. In *Proc. ICWN*, pages 109–113, 2003.
9. R. H. Güting, M. H. Böhlen, M. Erwig, C. S. Jensen, N. A. Lorentzos, M. Schneider, and M. Vazirgiannis. A Foundation for Representing and Quering Moving Objects. *ACM Trans. Database Syst.*, 25(1):1–42, 2000.
10. A. Guttman. R-Trees: A Dynamic Index Structure for Spatial Searching. In *Proc. SIGMOD*, pages 47–57, 1984.
11. J. Hightower and G. Borriello. Location Systems for Ubiquitous Computing. *IEEE Computer*, 34(8):57–66, 2001.
12. C. S. Jensen, D. Lin, and B. C. Ooi. Query and Update Efficient B$^+$-Tree Based Indexing of Moving Objects. In *Proc. VLDB*, pages 768–779, 2004.
13. G. Kollios, D. Gunopulos, and V. J. Tsotras. On Indexing Mobile Objects. In *Proc. PODS*, pages 261–272, 1999.
14. G. Kollios, V. J. Tsotras, D. Gunopulos, A. Delis, and M. Hadjieleftheriou. Indexing Animated Objects Using Spatiotemporal Access Methods. *IEEE Trans. Knowl. Data Eng.*, 13(5):758–777, 2001.
15. D. Lin, C. S. Jensen, B. C. Ooi, and S. Šaltenis. Efficient Indexing of the Historical, Present, and Future Positions of Moving Objects. In *Proc. MDM*, pages 59–66, 2005.
16. M. A. Nascimento and J. R. O. Silva. Towards Historical R-Trees. In *Proc. SAC*, pages 235–240, 1998.
17. D. Pfoser, C. S. Jensen, and Y. Theodoridis. Novel Approaches in Query Processing for Moving Object Trajectories. In *Proc. VLDB*, pages 395–406, 2000.
18. S. Šaltenis, C. S. Jensen, S. T. Leutenegger, and M. A. Lopez. Indexing the Positions of Continuously Moving Objects. In *Proc. SIGMOD*, pages 331–342, 2000.
19. M. Pelanis, S. Šaltenis, and C. S. Jensen. Indexing the Past, Present, and Anticipated Future Positions of Moving Objects. *ACM TODS*, 31(1):255–298, 2006.
20. Y. Tao and D. Papadias. MV3R-Tree: A Spatio-Temporal Access Method for Timestamp and Interval Queries. In *Proc. VLDB*, pages 431–440, 2001.
21. Y. Tao, D. Papadias, and J. Sun. The TPR*-Tree: An Optimized Spatio-Temporal Access Method for Predictive Queries. In *Proc. VLDB*, pages 790–801, 2003.
22. Y. Theodoridis, M. Vazirgiannis, and T. K. Sellis. Spatio-Temporal Indexing for Large Multimedia Applications. In *Proc. ICMCS*, pages 441–448, 1996.
23. R. Want. RFID Explained: A Primer on Radio Frequency Identification Technologies. *Synthesis Lectures on Mobile and Pervasive Computing*, 1(1):1–94, 2006.
24. O. Wolfson, B. Xu, S. Chamberlain, and L. Jiang. Moving Objects Databases: Issues and Solutions. In *Proc. SSDBM*, pages 111–122, 1998.
25. M. L. Yiu, Y. Tao, and N. Mamoulis. The B^{dual}-Tree: Indexing Moving Objects by Space Filling Curves in the Dual Space. *VLDBJ*, 17(3):379–400, 2008.