In Search of Indoor Dense Regions

An Approach Using Indoor Positioning Data

Li, Huan; Lu, Hua; Shou, Lidan; Chen, Gang; Chen, Ke

Published in:
IEEE Transactions on Knowledge and Data Engineering

DOI (link to publication from Publisher):
10.1109/TKDE.2018.2799215

Publication date:
2018

Document Version
Accepted author manuscript, peer reviewed version

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2018.2799215, IEEE Transactions on Knowledge and Data Engineering

In Search of Indoor Dense Regions: An Approach Using Indoor Positioning Data

Huan Li, Hua Lu*, *Senior Member, IEEE,* Lidan Shou, Gang Chen, and Ke Chen

Abstract—As people spend significant parts of daily lives indoors, it is useful and important to measure indoor densities and find the dense regions in many indoor scenarios like space management and security control. In this paper, we propose a data-driven approach that finds top-k indoor dense regions by using indoor positioning data. Such data is obtained by indoor positioning systems working at a relatively low frequency, and the reported locations in the data are discrete, from a preselected location set that does not continuously cover the entire indoor space. When a search is triggered, the object positioning information is already out-of-date and thus object locations are uncertain. To this end, we first integrate object location uncertainty into the definitions for counting objects in an indoor region and computing its density. Subsequently, we conduct a thorough analysis of the location uncertainty in the context of complex indoor topology, deriving upper and lower bounds of indoor region densities and introducing distance decaying effect into computing concrete indoor densities. Enabled by the uncertainty analysis outcomes, we design efficient search algorithms for solving the problem. Finally, we conduct extensive experimental studies on our proposals using synthetic and real data. The experimental results verify that the proposed search approach is efficient, scalable, and effective. The top-k indoor dense regions returned by our search are considerably consistent with the ground truth, despite that the search uses neither historical data nor extra knowledge about objects.

Index Terms—Indoor space, Indoor positioning data, Density queries

1 Introduction

Multiple studies [14], [26] disclose that people spend nearly 90% of their lives in indoor spaces such as office buildings, shopping malls, metro stations, and airports. As a result, many indoor spaces host large numbers of people even in a short period of time. For example, the New Town Plaza in Hong Kong saw weekend traffic up to 320,000 people in 2004 [31]. As the plaza usually opens from 11 AM to 10 PM [1], its hourly traffic was thus more than 29,000 on weekends. As another example, Beijing Capital International Airport's daily outbound passenger number was more than 260,000 in 2017 [2]. As there are fewer flights and passengers in late hours at night and there are also many inbound passengers, the peak hour passenger number must be tens of thousands in that airport. Moreover, as it is very usual for a person to spend more than an hour in a mall or an airport, the peak number of people at an instant of time should be of the same order of magnitude in the above examples.

When many people are inside an indoor space, it is useful and important to measure indoor densities and find the dense indoor regions in scenarios like space or flow management and security control. Considering a large airport, it is very useful for the authority to effectively determine which regions in a terminal are currently most crowded so that actions, e.g., to open more fast tracks, can be taken timely at the right place to help passengers in need. As another example, the security managers of a mall need to find the most crowded regions where currently there are high needs for extra services, e.g., more security guards for patrolling and diverging, in order to guarantee shoppers' safety.

At first thought, indoor density can be resolved by installing counting sensors at all the doors of each indoor partition, a basic topological unit (e.g., a room) connected to other units via doors. However, this naive approach in general is unattractive due to three reasons. First, it demands considerable investment on specialized hardware. Second, it entails appropriate redeployment of sensors whenever the indoor topology is changed, e.g., when a large conference hall is temporally divided into several smaller rooms or vice versa. Third, it is hardly possible to find a good position to install a sensor for a region in question that lacks doors or physical boundaries, e.g., open exhibition booths in a large exhibition hall.

In this research, we propose a low-cost, data-driven approach to the indoor density that harvests needed values from indoor positioning data. Such data is increasingly available, thanks to the advance in indoor positioning [1] and the high smartphone penetration in many countries[3]. In contrast to the naive approach, our approach demands no specific hardware to be installed, it is not bothered by indoor topology variations, and it works well for user-defined indoor regions.

Assuming GPS data or the like, techniques for outdoor density queries in Euclidean spaces [8], [9], [13], [15], [25] or road networks [18], [20], [34] fall short in indoor settings. Indoor spaces feature distinct entities such as rooms, walls, doors and staircases, which altogether form complex indoor topology that enables and constrains movements. Therefore, indoor spaces are modeled [4], [16], [19], [23], [28], [29] differently from outdoor Euclidean spaces or road networks. We must take into account the particularities of indoor topology appropriately when computing indoor densities. Unlike GPS that reports longitude and latitude continuously, indoor positioning systems [7] offer lower

- *H. Li, L. Shou, G. Chen, and K. Chen are with Department of Computer Science, Zhejiang University, China. E-mail: {lihuancs, should, cg, chenk}@zju.edu.cn*
- *H. Lu is with the Department of Computer Science, Aalborg University, Denmark. E-mail: luhua@cs.aau.dk*

H. Lu is the corresponding author.

1. https://goo.gl/1fR5Xo
2. https://goo.gl/MTaE95
3. http://goo.gl/pdtvMM

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2018.2799215, IEEE Transactions on Knowledge and Data Engineering

sampling frequency and only report discrete indoor locations (see Section 2.1 for more details), thus leaving considerable uncertainty in indoor positioning data to be used to compute indoor densities. This issue becomes even more challenging to deal with in the context of complex indoor topology. Thus, we must handle indoor positioning data appropriately in order to find indoor dense regions effectively and efficiently. Also, many indoor spaces host large numbers of people as mentioned above. Therefore, computing indoor densities is not a trivial task.

We formulate the problem of finding top-k indoor dense regions by using indoor positioning data. In our setting, a user is allowed to customize the set of query regions from which the k densest ones are returned. To enable the search, the latest positioning information for indoor moving objects is maintained in an online indoor positioning table ($OIPT$), where a record (o, loc, t) means the object o was reported to be at location loc at a past time t and there is no more recent information available from the indoor positioning system.

When a search is triggered at the current time, the object positioning information in $OIPT$ is already out-of-date due to the relatively low frequency and discrete nature of indoor positioning, and thus object locations are uncertain—an object's current location is within an uncertainty region. Due to the location uncertainty, directly counting the objects in an indoor region does not work for computing the region's density. To this end, we integrate object uncertainty regions into the definitions for counting objects in an indoor region and computing its density. Subsequently, we propose the novel concepts of indoor buffer region and indoor core region to derive the upper and lower bounds of a region's density. By using the density bounds, the density computation is able to concentrate on the relevant objects only and thus can be done more efficiently. It is non-trivial to compute indoor buffer (core) regions because they involve not only location uncertainty but also complex indoor topology. Based on the indoor buffer (core) regions, we derive effective upper (lower) bounds for indoor densities that enable aggressive data pruning in the top-k search. When computing concrete indoor densities, we consider the realistic distance decaying effect instead of assuming a uniform distribution of an object's possible locations in its uncertainty region. Moreover, we design search algorithms for finding the current top-k indoor dense regions using the upper and lower bounds. Experimental studies on synthetic and real data sets demonstrate that our search algorithms are efficient, scalable and effective. The returned top-k indoor dense regions are significantly consistent with ground truth although only the most recent indoor positioning data is used in the search.

We make the following contributions in this paper.

- We design an indoor density definition amenable to indoor object location uncertainty, and formulate the problem of finding top-k indoor dense regions.
- We analyze the indoor location uncertainty, derive bounds of indoor densities, and introduce distance decaying effect into indoor density computing.
- By making use of the uncertainty analysis outcomes, we design efficient algorithms to search for the current top-k indoor dense regions.
- We conduct extensive experimental studies to evaluate our proposals on synthetic and real data sets.

The paper is organized as follows. Section 2 formulates the problem. Section 3 gives the uncertainty analysis. Section 4 details the search algorithms. Section 5 reports on the experimental studies. Section 6 reviews literature. Section 7 concludes and discusses future work.

2 Preliminaries

Table 1 lists the notations used throughout this paper.

Table 1
Notations

Symbol	Meaning
o_i	An indoor moving object
O	The set of all indoor moving objects
r, r_i	Indoor regions
Q	The set of indoor regions in a query
t_c	The current time when a query is issued
t_{min}	$\min\{rec.t \mid rec \in OIPT\}$
Δt	$t_c - t_{min}$
$UR_I(loc)$	The indoor uncertainty region of object location loc
$\tau_O(r)$	Indoor region r's density with respect to O
$\Theta_I^{\triangleright}(r)$	Indoor region r's indoor buffer region
$\Theta_I^{\triangleleft}(r)$	Indoor region r's indoor core region

2.1 Indoor Positioning Data

In our indoor setting, a positioning system emits positioning reports in the format of $(objectID, loc,\ t)$, where $objectID$ identifies an object, loc is an indoor location and t is a time stamp, meaning the object's location is estimated to be loc at time t. In our abstraction, a location loc is generic in that it can either be a point or a small (circular) region. It is able to model data obtained by different positioning approaches. For example, in fingerprinting based Wi-Fi indoor positioning [11], a loc is one of the preselected reference locations in an indoor space. In proximity based indoor positioning [10], a loc is the detection range of a sensor like RFID reader. For the sake of simplicity and conciseness, we use a point to represent loc in our solution which nevertheless can also accommodate a region representation of loc.

Due to limited storage and low throughput, an indoor positioning system may choose not to store historical data. Therefore, we focus on online indoor positioning data without using historical data. Our experimental studies show that our solution using online data only is able to find the top-k indoor dense regions considerably consistent with ground truth. In particular, we only keep the latest report for an object and store such reports for all objects in an *online indoor positioning table* ($OIPT$), as exemplified in Table 2.

Table 2
Example of $OIPT$

objectID	location	t
o_1	l_1	t_1
o_2	l_3	t_1
o_3	l_6	t_6

The positioning system produces such reports aperiodically for an object, and therefore the time stamps in $OIPT$ are unnecessarily the same across objects. We use t_{min} to denote the minimum time stamp in $OIPT$, i.e., $t_{min} = \min\{rec.t \mid rec \in OIPT\}$. In the example shown in Table 2, $t_{min} = t_1$. Further, V_{max} denotes the maximum speed of all objects moving in the indoor space under discussion.

It is noteworthy that the indoor positioning data is of low accuracy [21] compared to outdoor GPS. For example, fingerprinting based indoor positioning only snaps to a fixed set of preselected indoor locations when making location estimates. The proximity based indoor positioning does not report object presence if it is outside of any sensor's detection range. Thus, indoor positioning and the data generated are not continuous but discrete. This is a significant distinction compared to outdoor GPS where the location reports are in nature continuous as longitude and latitude coordinates.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2018.2799215, IEEE Transactions on Knowledge and Data Engineering

2.2 Problem Formulation

Our problem formulation starts with indoor regions. Basically, an *indoor region*[4] refers to a continuous part of indoor space covered by a geometric shape. For simplicity, we use rectangles to represent indoor regions when the context is clear throughout the paper. Nevertheless, our definitions and relevant techniques support arbitrary shapes.

In a certain data context where we know each object's exact location when a density query is issued at time t_c, the density of a region r is simply the number of objects in r divided by r's area. However, this simple calculation does not work in our setting. The difficulty is mainly due to the difference between objects' positioning time and the query time t_c. For any record $rec \in OIPT$, we have $rec.t \leq t_c$. In general, the reported locations in $OIPT$ are very likely to be already out-of-date and the objects may have left their reported locations when the query is issued at time t_c.

The discrete nature of indoor positioning (see Section 2.1) renders it impossible to know an object's exact position at an arbitrary query time t_c. We derive an object's uncertain whereabout at query time t_c and quantify indoor region density as follows. Given an indoor location loc reported at a latest sampling time t_l, its indoor uncertainty region $UR_I(loc, t_c, t_l)$ describes the indoor portions where the object can reach at the current time t_c under the maximum speed constraint V_{max}. When the time context is clear we use $UR_I(loc)$ for simplicity. Formally, $UR_I(loc) = Range_I(loc, V_{max} \cdot (t_c - t_l))$.[5] That said, $UR_I(loc)$ consists of the indoor portions that are within the indoor distance $V_{max} \cdot (t_c - t_l)$ from indoor location loc. Note that this distance is the maximum distance an object can move from time t_l to time t_c.

Example 1. In Figure 1, $UR_I(l_1)$ for location l_1 is the circular region fully inside the indoor space, whereas $UR_I(l_2)$ for location l_2 is the shaded part of such a circular region.

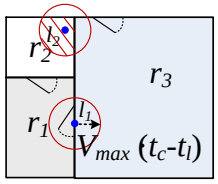


Figure 1. Indoor Uncertainty Regions

Accordingly, we define *object presence* that stipulates how an object should be counted for an indoor region r at time t_c.

Definition 1 (Object Presence). For an indoor object o with a record (o, loc, t) in $OIPT$, o's *presence* in a region r is defined as $\phi_r(o) = \frac{Area(UR_I(loc) \cap r)}{Area(UR_I(loc))}$.

Object o's presence $\phi_r(o)$ is zero if its $UR_I(loc)$ does not overlap region r. Clearly, $\phi_r(o) \leq 1$ always holds for an arbitrary region r and an arbitrary object o.

So far we have assumed that the object's possible location in a given uncertainty region $UR_I(loc)$ follows a uniform distribution. However, this straightforward assumption does not capture the reality well. A more realistic consideration is that the object is more likely to be close to the reported location loc. In other words, the probability that the object is located at $loc' \in UR_I(loc)$ at time t_c decreases if loc' is farther away from loc. Such a distance decaying effect reflects the locality of object movement and it is consistent with the observed evidence in human geography [5].

4. The space context throughout this paper is indoor unless it is explicitly stated otherwise. Therefore, term *region* alone may refer to an indoor region when the context is clear.

5. Indoor range query $Range_I(l, \delta)$ returns *all* indoor portions within the indoor distance δ from location l [23]; the search may involve multiple doors.

There exist multiple functions to characterize the distance decaying effect for an uncertainty region $UR_I(loc)$. The computations of $UR_I(loc)$'s area and its intersection with an indoor query region r depend on the concrete definition of such a distance decaying function. We give the generic definitions in this section and detail the distance decaying functions and pertinent object presence computation in Section 3.3.

With the concept of object presence, we define *load* that "counts" the objects for an indoor region.

Definition 2 (Load). Given a set O of indoor moving objects, a region r's *load* is defined as $\lambda_O(r) = \sum_{o \in O} \phi_r(o)$.

Let $m \leq |O|$ be the number of objects whose uncertainty regions overlap region r. Note that $\lambda_O(r) \leq m$ always holds, because each object $o \in O$ contributes at most 1 to $\lambda_O(r)$.

Now we are ready to define *density* for an indoor region.

Definition 3 (Density). Given a set O of indoor moving objects, an indoor region r's density is $\tau_O(r) = \frac{\lambda_O(r)}{Area(r)}$.

Problem 1 (Top-k Indoor Dense Region Search). Given an online set O of indoor moving objects, whose latest positioning records are captured in an $OIPT$, and a query set Q of indoor regions, the top-k indoor dense region search returns k densest indoor regions in a k-subset $Q_k \subseteq Q$ such that $\forall r \in Q_k, \forall r' \in Q \setminus Q_k, \tau_O(r) \geq \tau_O(r')$.

Our problem formulation does not restrict query regions in set Q to be indoor partitions. As users like building officers usually know well what kinds of query regions to be included in Q, our top-k search allows users to customize semantic-dependent query regions according to their practical needs. For example, a query region in Q can be defined as a part of an indoor partition or a combination of several partitions, as to be discussed in Section 3.1.3. This flexibility renders the top-k indoor dense region search very useful in special cases, e.g., to find the densest zones in a large, open exhibition hall that is a single indoor partition.

3 Uncertainty Analysis

This section presents a thorough analysis of the uncertainties involved in our research problem. In Section 3.1, we propose the concepts of indoor buffer region and indoor core region to address the uncertainty regions associated with indoor objects. The concepts are used to derive bounds of indoor region density in Section 3.2. In Section 3.3, we introduce distance decaying to uncertainty regions.

3.1 Indoor Buffer Region and Indoor Core Region

3.1.1 Indoor Buffer Region

In the context of free-moving objects, suppose that an object's last (or latest) positioning report $(objectID, loc, t_l)$ is received at a past time t_l. At the current time t_c, the object may be still at the location loc, or has left. This results in object location uncertainty—the object's current location is constrained by a circular region centered at loc with a radius $V_{max} \cdot (t_c - t_l)$ where V_{max} is the object's maximum possible speed. Due to the object location uncertainty, a region r where there was no object at a past time can contain objects at the current time. However, such objects cannot come from locations too far away due to the maximum speed constraint. Instead, they can only come from an extended region that contains region r. We call such an extended

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2018.2799215, IEEE Transactions on Knowledge and Data Engineering

region *general buffer region* or simply *buffer region*, and give its definition first in the general context below.

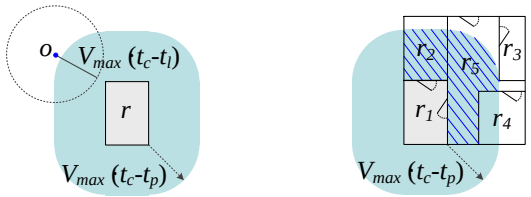
Definition 4 (Buffer Region). Given a past time t_p, the current time t_c ($t_c \geq t_p$), and the maximum speed constraint V_{max} for moving objects, a region r's *buffer region* $\Theta^{\triangleright}(r, t_c, t_p)$ is a δ-*Minkowski region* [6] where $\delta = V_{max} \cdot (t_c - t_p)$.

In our analysis, both t_p and t_l represent past time but they carry different meanings. Put it simply, t_l is always associated with a particular object whereas t_p is not. In this sense, a buffer region $\Theta^{\triangleright}(r, t_c, t_p)$ is generic and independent of particular objects. If we know an object's t_l, we can infer whether the object is currently in $\Theta^{\triangleright}(r, t_c, t_p)$ or not.

Example 2. An example of buffer region is shown in Figure 2(a). Let object o's latest position report be (o, loc, t_l). If $loc \notin \Theta^{\triangleright}(r, t_c, t_p)$ and $t_l > t_p$, i.e., object o was last seen outside $\Theta^{\triangleright}(r, t_c, t_p)$ at time t_l that is later than t_p, then object o cannot be in region r at the current time t_c.

Next, we define *indoor buffer region* as follows.

Definition 5 (Indoor Buffer Region). Given a past time t_p, the current time t_c, and the maximum speed constraint V_{max}, an indoor region r's *indoor buffer region* $\Theta_I^{\triangleright}(r, t_c, t_p)$ is composed of indoor portions fully contained by r's buffer region $\Theta^{\triangleright}(r, t_c, t_p)$, and the shortest indoor path from any point in such a portion to r is fully contained by $\Theta^{\triangleright}(r, t_c, t_p)$.



(a) General Buffer Region (b) Indoor Buffer Region

Figure 2. Buffer Regions

Generally speaking, r's indoor buffer region $\Theta_I^{\triangleright}(r, t_c, t_p)$ is the intersection of r's buffer region $\Theta^{\triangleright}(r, t_c, t_p)$ and those indoor parts from where one can reach r within time interval $[t_p, t_c]$.

3.1.2 Indoor Core Region

Opposite to indoor buffer regions, we define an indoor region r's *indoor core region* as a reduced region of r from where one cannot leave r within time interval $[t_p, t_c]$. The definition is given as follows.

Definition 6 (Indoor Core Region). Given a past time t_p, the current time t_c, and the maximum speed constraint V_{max}, an indoor region r's *indoor core region* $\Theta_I^{\triangleleft}(r, t_c, t_p)$ is the portion of r such that the shortest indoor path from any point in the portion to any of r's doors is no less than $V_{max} \cdot (t_c - t_p)$.

If we know an object was seen in $\Theta_I^{\triangleleft}(r, t_c, t_p)$ at a past time t_l after t_p, i.e., $t_l > t_p$, we can infer that it cannot reach any door of r and thus is still inside r at the current time t_c.

When the time stamps are not of particular interest, we simply use $\Theta_I^{\triangleright}(r)$ and $\Theta_I^{\triangleleft}(r)$ to denote r's indoor buffer region and indoor core region, respectively. Next, we present how to determine $\Theta_I^{\triangleright}(r, t_c, t_p)$ and $\Theta_I^{\triangleleft}(r, t_c, t_p)$ for an indoor region r.

3.1.3 Determination

Lemmas 1 and 2 quickly prune irrelevant portions when we determine an indoor buffer region.

Lemma 1. Given two indoor regions r_1 and r_2, if r_2 is outside r_1's general buffer region $\Theta^{\triangleright}(r_1, t_c, t_p)$, r_2 cannot be a part of r_1's indoor buffer region, i.e., $r_2 \notin \Theta_I^{\triangleright}(r_1, t_c, t_p)$.

Proof 1. Note that $\Theta_I^{\triangleright}(r_1, t_c, t_p) \subseteq \Theta^{\triangleright}(r_1, t_c, t_p)$, therefore $r_2 \notin \Theta_I^{\triangleright}(r_1, t_c, t_p)$ if $r_2 \notin \Theta^{\triangleright}(r_1, t_c, t_p)$.

Lemma 2. For indoor regions r_1 and r_2, if all r_2's doors are outside r_1's general buffer region $\Theta^{\triangleright}(r_1, t_c, t_p)$, r_2 cannot be a part of r_1's indoor buffer region, i.e., $r_2 \notin \Theta_I^{\triangleright}(r_1, t_c, t_p)$.

Proof 2. (Sketch) If all r_2's doors are outside $\Theta^{\triangleright}(r_1, t_c, t_p)$, they must be outside $\Theta_I^{\triangleright}(r_1, t_c, t_p)$ too (Lemma 1). For any point pt in r_2 to reach point pt' in room r_1, it must go through one of r_2's doors, say dr. As dr is already outside $\Theta_I^{\triangleright}(r_1, t_c, t_p)$, pt cannot reach pt' via dr by time t_c.

Example 3. An example is given in Figure 2(b). When we consider region r_1's indoor buffer region, we first exclude room r_3 according to Lemma 1 since it is outside r_1's general buffer region. Furthermore, we also exclude room r_4 according to Lemma 2 although r_4 overlaps r_1's general buffer region. A moving object cannot reach r_1 from r_4 within the time length $t_c - t_p$, as r_4's only door is outside $\Theta^{\triangleright}(r_1, t_c, t_p)$. In contrast, we include the intersection part of $\Theta^{\triangleright}(r_1, t_c, t_p)$ and room r_2 since r_2 is connected to r_1 by a door fully inside $\Theta^{\triangleright}(r_1, t_c, t_p)$.

In the example above, excluding rooms r_3, r_4, and part of r_2 and r_5 gives us the shaded portion as shown in Figure 2(b). However, this does not give r_1's precise indoor buffer region. Actually, region r_1's indoor buffer region $\Theta_I^{\triangleright}(r_1, t_c, t_p)$ is the union of room r_1 and *part* of the shaded portion in the figure.

Next, we elaborate on which parts of the shaded portion should be included in $\Theta_I^{\triangleright}(r_1, t_c, t_p)$. We assume that all doors are bidirectional. Nevertheless, the techniques proposed in this paper can be extended to handle unidirectional doors. We use $P2D(r)$ [23] to get the set of doors associated to an indoor partition r. If a given query region r is *exactly an indoor partition*, Lemma 3 tells that it is sufficient to consider r's doors when we determine its indoor buffer region $\Theta_I^{\triangleright}(r_1, t_c, t_p)$.

Lemma 3. An indoor partition r's indoor buffer region is the union of r and all indoor portions within indoor distance $\delta = V_{max} \cdot (t_c - t_p)$ from each door associated to r. Formally, $\Theta_I^{\triangleright}(r, t_c, t_p) = r \cup \bigcup_{d \in P2D(r)} Range_I(d, \delta)$.

Proof 3. (Sketch) This lemma can be proved by Lemmas 1 and 2, and the fact that an indoor distance cannot be shorter than the corresponding Euclidean distance.

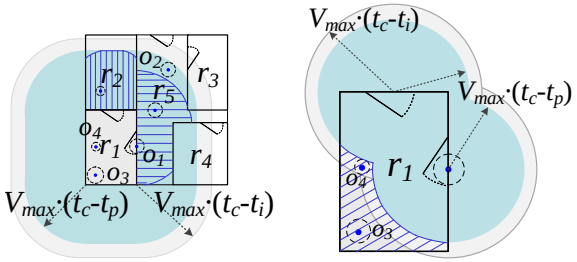
Given an indoor partition r, Lemma 4 utilizes the shortest indoor path to r's doors to determine its indoor core region $\Theta_I^{\triangleleft}(r_1, t_c, t_p)$.

Lemma 4. An indoor partition r's indoor core region is the part of r that excludes all indoor portions within indoor distance $\delta = V_{max} \cdot (t_c - t_p)$ from each door associated to r. Formally, $\Theta_I^{\triangleleft}(r, t_c, t_p) = r \setminus \bigcup_{d \in P2D(r)} Range_I(d, \delta)$.

Proof 4. (Sketch) For any point $pt \in r$ and any door $d \in P2D(r)$, if $pt \notin \bigcup_{d \in P2D(r)} Range_I(d, \delta)$, pt cannot reach d within $t_c - t_p$ and is still contained by r at time t_c.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2018.2799215, IEEE Transactions on Knowledge and Data Engineering

The precise indoor buffer region for room r_1 in Figure 2(b) is illustrated as the union of r_1 and the shaded parts in Figure 3(a). The precise indoor core region of r_1 is illustrated as the shaded part in Figure 3(b).



(a) Indoor Buffer Region (b) Indoor Core Region

Figure 3. Precise Indoor Buffer Region and Indoor Core Region

On the other hand, an indoor region r in a query may not be equal to an indoor partition. Instead, a query region r may be a combination of complete and/or incomplete indoor partitions. Our approach supports user-defined shapes of indoor query regions. For the simplicity of presentation, we consider a rectangular indoor region r. A side of r can be: i) a wall without a door, ii) a wall with a door(s), iii) an open segment fully inside an indoor partition, or iv) combination of i) to iii).

Example 4. Figure 4 illustrates a complex query region r_a. It is the shaded region composed of a part of room r_1, a part of room r_5, and the whole room r_4. Its left, right and bottom sides are all walls without doors, whereas its top side consists of three parts: an open segment inside room r_1, an open segment inside room r_5, and a wall with a door from room r_4.

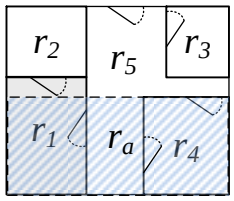


Figure 4. Example of Complex Query Region

Algorithm 1 determines an arbitrary region r's indoor buffer and core regions. Here, δ is still $V_{max} \cdot (t_c - t_p)$. For r's each door d (line 3), variable r_m that contains the indoor portion within indoor distance δ from d (line 4) is included for indoor buffer region $\Theta_I^{\triangleright}$ and excluded from indoor core region $\Theta_I^{\triangleleft}$, respectively (line 5). For r's each open segment g (line 6), let r_m be the intersection of g's δ-Minkowski region ($\mathcal{M}(g, \delta)$ in line 8) and g's containing indoor partition p (line 9). Then r_m is included for $\Theta_I^{\triangleright}$ and excluded from $\Theta_I^{\triangleleft}$ (line 10). Expansion for determining $\Theta_I^{\triangleright}$ is done further outwardly through all accessible doors within the range of general buffer region $\Theta^{\triangleright}(g)$ (lines 11–13).

Algorithm 1 DetermineIbcRs(Region r, Distance δ)

1: $\Theta_I^{\triangleright} \leftarrow r$; $\Theta_I^{\triangleleft} \leftarrow r$
2: **for** r's each side λ **do**
3: **for** each door d on λ **do**
4: $r_m \leftarrow Range_I(d, \delta)$
5: $\Theta_I^{\triangleright} \leftarrow \Theta_I^{\triangleright} \cup r_m$; $\Theta_I^{\triangleleft} \leftarrow \Theta_I^{\triangleleft} \setminus r_m$
6: **for** each open segment g on λ **do**
7: find the indoor partition p that contains g
8: get g's general buffer region $\Theta^{\triangleright}(g) \leftarrow \mathcal{M}(g, \delta)$
9: $r_m \leftarrow p \cap \Theta^{\triangleright}(g)$
10: $\Theta_I^{\triangleright} \leftarrow \Theta_I^{\triangleright} \cup r_m$; $\Theta_I^{\triangleleft} \leftarrow \Theta_I^{\triangleleft} \setminus r_m$
11: **for** each door $d \in P2D(p)$ and in $\Theta^{\triangleright}(g)$ **do**
12: get the shortest indoor distance δ' from d to g
13: $\Theta_I^{\triangleright} \leftarrow \Theta_I^{\triangleright} \cup Range_I(d, \delta - \delta')$
14: **return** $\Theta_I^{\triangleright}$, $\Theta_I^{\triangleleft}$

3.2 Bounds of Indoor Density

The inherent uncertainty in indoor positioning data renders it complex to compute indoor densities. Therefore, we derive upper and lower bounds of indoor densities by making use of the concepts of indoor buffer and core regions detailed in Sections 3.1.

We use function $COUNT(r)$ to obtain the number of objects whose last reported location is contained by an indoor region r. The following lemma gives the upper and lower bounds (*ULB* in short) that involve moving objects within r's indoor buffer region $\Theta_I^{\triangleright}(r)$ and indoor core region $\Theta_I^{\triangleleft}(r)$.

Lemma 5 (Indoor Density ULB). $\frac{COUNT(\Theta_I^{\triangleleft}(r))}{Area(r)} \leq \tau_O(r) \leq \frac{COUNT(\Theta_I^{\triangleright}(r))}{Area(r)}$.

Proof 5. The ULB is equivalent to $COUNT(\Theta_I^{\triangleleft}(r)) \leq \lambda_O(r) \leq COUNT(\Theta_I^{\triangleright}(r))$. As each object o last seen in $\Theta_I^{\triangleleft}(r)$ cannot leave r by the current time, such an o's presence in region r must be 1. Based on Definition 2, we have $COUNT(\Theta_I^{\triangleleft}(r)) \leq \lambda_O(r)$. On the other hand, for any object o last seen at time t_l no later than t_p, i.e., $t_l \geq t_p$, if o has contributed to $\lambda_O(r)$, its shortest indoor distance to region r must be shorter than $V_{max} \cdot (t_c - t_l)$. Based on Definition 5, such an o must be inside $\Theta_I^{\triangleright}(r)$ at time t_l and thus we have $\lambda_O(r) \leq COUNT(\Theta_I^{\triangleright}(r))$. So the lemma is proved.

Lemma 6 loosens the indoor density ULB by using a longer temporal interval that ends with the current time t_c.

Lemma 6 (Temporal Loose ULB). For two past time stamps t_i and t_p, if $t_i \leq t_p$ we have $\frac{COUNT(\Theta_I^{\triangleleft}(r,t_c,t_i))}{Area(r)} \leq \frac{COUNT(\Theta_I^{\triangleleft}(r,t_c,t_p))}{Area(r)} \leq \tau_O(r) \leq \frac{COUNT(\Theta_I^{\triangleright}(r,t_c,t_p))}{Area(r)} \leq \frac{COUNT(\Theta_I^{\triangleright}(r,t_c,t_i))}{Area(r)}$.

Proof 6. Since t_i is older than t_p, $\Theta_I^{\triangleright}(r, t_c, t_i)$ expands further away from region r and $\Theta_I^{\triangleright}(r, t_c, t_i)$ is larger than $\Theta_I^{\triangleright}(r, t_c, t_p)$. Likewise, $\Theta_I^{\triangleleft}(r, t_c, t_i)$ shrinks more inwardly and is smaller than $\Theta_I^{\triangleleft}(r, t_c, t_p)$. As a result, we have the following inequalities $\lambda_O(r) \leq COUNT(\Theta_I^{\triangleright}(r, t_c, t_p)) \leq COUNT(\Theta_I^{\triangleright}(r, t_c, t_i))$ and $COUNT(\Theta_I^{\triangleleft}(r, t_c, t_i)) \leq COUNT(\Theta_I^{\triangleleft}(r, t_c, t_p)) \leq \lambda_O(r)$. So the lemma holds.

Example 5. In Figure 3(a), there are six moving objects and their most recent location reports and uncertainty regions are indicated by small dots and dashed circles, respectively. Suppose that the drawing precisely reflects the situation at the current time t_c, and room r_1's area is ten square meters. There are two objects o_3 and o_4 whose uncertainty regions are fully in r_1, and half of object o_1's is in r_1. Therefore, r_1's density is $\tau_O(r_1) = 2.5/10 = 0.25$. Moreover, r_1's indoor buffer region $\Theta_I^{\triangleright}(r_1, t_c, t_p)$ is the union of r_1 and the shaded parts. As a result, the indoor density upper bound is $COUNT(\Theta_I^{\triangleright}(r_1, t_c, t_p))/Area(r_1) = 5/10 = 0.5 > \tau_O(r_1)$. Indicated by the outer rounded rectangle $\Theta^{\triangleright}(r_1, t_c, t_i)$ in Figure 3(a), r_1's indoor buffer region $\Theta_I^{\triangleright}(r_1, t_c, t_i)$ is larger and contains o_2's reported location. Thus, we have the temporal loose upper bound as $COUNT(\Theta_I^{\triangleright}(r_1, t_c, t_i))/Area(r_1) = 6/10 = 0.6 > COUNT(\Theta_I^{\triangleright}(r_1, t_c, t_p))/Area(r_1) = 0.5 > \tau_O(r_1) = 0.25$. Take a close look at r_1 in Figure 3(b), the indoor

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2018.2799215, IEEE Transactions on Knowledge and Data Engineering

density lower bound indicated by r_1's indoor core region is $COUNT(\Theta_I^{\triangleleft}(r_1, t_c, t_p))/Area(r_1) = 2/10 = 0.2 < \tau_O(r_1)$. Also, the smaller indoor core region $\Theta_I^{\triangleleft}(r_1, t_c, t_i)$ excludes o_4's reported location, thus we have the temporal loose lower bound as $COUNT(\Theta_I^{\triangleleft}(r_1, t_c, t_i))/Area(r_1) = 1/10 = 0.1 < COUNT(\Theta_I^{\triangleleft}(r_1, t_c, t_p))/Area(r_1) = 0.2$.

3.3 Distance Decaying in Uncertainty Regions

As discussed in Section 2.2, *distance decaying* is a phenomenon commonly observed in object movements—the farther a destination location is, the less likely an object moves to it from its current location. Such a decaying effect is formally characterized by a *distance decaying function* (DDF).

Definition 7 (Distance Decaying Function). Given a reported location loc, an arbitrary location l, and their indoor distance $\delta = dist_I(loc, l)$, a *distance decaying function (DDF)* $\Gamma(\delta)$ decreases as the indoor distance δ increases.

Example 6. A DDF usually is a monotone nonincreasing function. Table 3 lists commonly used DDFs. *Linear Decay Law* (LDL) describes the functions that decrease linearly as the variable increases. Function $\Gamma(\delta) = 1 - \delta/D$ is a basic form of LDL, where $D = \max_{l \in UR_I(loc)} dist_I(loc, l)$ is the maximum indoor distance that an object can move from loc to an arbitrary location l inside $UR_I(loc)$. *Inverse 1st Power Law* (I1PL), *Inverse 2nd Power Law* (I2PL) and *Exponential Decay Law* (EDL) are three typical non-linear decaying functions that decrease rapidly with increasing variables. The Constant Law (CL) function $\Gamma(\delta) = C$ is insensitive to the variable and always returns a constant C. CL assumes that object locations follow a uniform distribution in $UR_I(loc)$.

Table 3
Example of DDFs

Notation	Basic Form	Law Name
LDL	$\Gamma(\delta) = 1 - \delta/D$	Linear Decay Law
I1PL	$\Gamma(\delta) = 1/(\delta + 1)$	Inverse 1st Power Law
I2PL	$\Gamma(\delta) = 1/(\delta + 1)^2$	Inverse 2nd Power Law
EDL	$\Gamma(\delta) = e^{-\delta}$	Exponential Decay Law
CL	$\Gamma(\delta) = C$	Constant Law

A DDF $\Gamma(\delta)$ differentiates an object's presence at each possible location inside $UR_I(loc)$. Accordingly, we define the distance decaying version of the object presence (c.f. Definition 1).

Definition 8 (Distance Decaying Object Presence). Given an indoor query region r, an object o's indoor uncertainty region $UR_I(loc)$ with the distance decaying function Γ, o's presence in r is rewritten as $\phi_r^{\Gamma}(o) = \frac{\int_{l \in (UR_I(loc) \cap r)} \Gamma(dist_I(loc,l))dl}{\int_{l \in UR_I(loc)} \Gamma(dist_I(loc,l))dl}$.

As $\phi_r^{\Gamma}(o) \leq 1$, the ULBs derived in Section 3.2 still hold when $\phi_r^{\Gamma}(o)$ is used for computing indoor density.

Discussion Our uncertainty analysis presented in this section is orthogonal to the detailed shapes of indoor regions and buffer (core) regions. Algorithm 1 can be adapted to handle an arbitrary indoor region shape given that its geometry and the door locations are specified. The density ULBs work for any shapes whose areas can be calculated. Hence, the search algorithms to be detailed in Section 4 are also independent of the concrete shapes. In our implementation, we employ integration techniques to handle the geometry related computations, e.g., the areas and intersections of complex indoor regions, and the distance decaying related computations. Briefly, an indoor (uncertainty) region is divided into a number of primitive shapes like circles, triangles, and/or sectors, to which the integral equation method is then applied. We omit the low-level details here as they are not the focus of our research.

4 Algorithms for Top-k Indoor Dense Region Search

This section details the algorithms that search for the current top-k indoor dense regions. Section 4.1 gives the overall framework for the search. Sections 4.2 and 4.3 propose a one-pass search method and an improved method, respectively. Section 4.4 compares the costs of the two methods.

4.1 Overall Framework

We use an R-tree R_P to index indoor partitions. For each partition, a bucket contains the IDs of all objects whose latest reported location in $OIPT$ overlaps that partition. The $OIPT$ is organized as a hash table with object IDs as the key. It is noteworthy that the R-tree R_P itself is constructed only once, i.e., it is not reconstructed for every query since there is no change on the indoor partitions. In our implementation, we use another thread to update the $OIPT$ and the object ID buckets corresponding to indoor partitions, which runs in parallel to the query processing thread. Upon receiving a positioning report of an object, the updating thread checks if the object has moved to another indoor partition since its last positioning report was recorded, and moves the object ID to the new corresponding bucket if necessary.

Algorithm 2 is the overall framework for finding the top-k indoor dense regions. It uses a max-heap H to control the processing order of indoor regions in the query set (line 1), and a hash table h_Q (line 2) to maintain the set of objects involved in the density computation for a query region r. More importantly, it makes use of the following pruning rule that is enabled by Lemmas 5 and 6 in Section 3.

Pruning Rule 1 (Bound Pruning). Given two indoor regions r_1 and r_2, the following two properties hold:

1) If $LowerBound(\tau_O(r_1)) > UpperBound(\tau_O(r_2))$, then $\tau_O(r_1) > \tau_O(r_2)$. Moreover, if r_1's lower bound density $LowerBound(\tau_O(r_1))$ is the k-th biggest in all regions, region r_2 can be safely pruned without having its density $\tau_O(r_2)$ calculated.
2) If $\tau_O(r_1) > UpperBound(\tau_O(r_2))$, then $\tau_O(r_1) > \tau_O(r_2)$. If r_1's density $\tau_O(r_1)$ is the lowest in the top-k results so far, region r_2 can also be safely pruned.

Specifically, Algorithm 2 overestimates as well as underestimates a query region r's density $\tau_O(r)$ both spatially and temporally. According to Lemma 6, we get the oldest time stamp $t_{min} = \min\{rec.t \mid rec \in OIPT\}$ in $OIPT$, and use t_{min} to derive region r's indoor buffer region ibr and indoor core region icr (lines 4–7). According to Lemma 5, we overestimate (underestimate) r's load by counting all the objects whose last reported location is in ibr (icr) (line 8 calling Algorithm 3). All other objects are pruned due to the property of an indoor buffer region. Subsequently, Algorithm 2 stores the overestimated object set $set_{\top}$ as well as the underestimated object set $set_{\perp}$ in the hash table h_Q for a query region r (line 9). Also, each lower bound density is calculated as $\frac{|set_{\perp}|}{Area(r)}$ and added to the set $S_{\perp}$ (lines 3 and 10). After that, according to the first property in Pruning

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2018.2799215, IEEE Transactions on Knowledge and Data Engineering

Rule 1, only the regions whose upper bound density $\frac{|set_\top|}{Area(r)}$ is no less than the k-th biggest lower bound density in $S_\perp$ should be further processed (lines 11–14). For each such r, a flag value OE_IBR, and r's upper bound density are pushed as a single element into a max-heap H (line 15). The flag value OE_IBR is to indicate that r's associated density is an overestimate from its indoor buffer region. This information is needed for subsequent process, to be detailed in Sections 4.2 and 4.3.

Algorithm 2 TopkIDRs(Indoor query region set Q, Partition R-tree R_P, Online indoor positioning table $OIPT$, Current time t_c)

1: initialize a max-heap H
2: initialize a hash table $h_Q : Q \rightarrow \{(2^{ObjectID}, 2^{ObjectID})\}$
3: initialize a lower bound density set $S_\perp$
4: $t_{min} \leftarrow \min\{rec.t \mid rec \in OIPT\}$
5: $\delta \leftarrow V_{max} \cdot (t_c - t_{min})$
6: **for** each region $r \in Q$ **do**
7: $\quad ibr, icr \leftarrow$ DetermineIbcRs(r, δ)
8: $\quad (set_\top, set_\perp) \leftarrow$ COUNT4ibcRs(ibr, icr, R_P)
9: $\quad h_Q[r] \leftarrow (set_\top, set_\perp)$
10: $\quad$ add $\frac{|set_\perp|}{Area(r)}$ to $S_\perp$
11: $kbound \leftarrow$ the k-th biggest in $S_\perp$
12: **for** each region $r \in Q$ **do**
13: $\quad (set_\top, set_\perp) \leftarrow h_Q[r]$
14: $\quad$ **if** $\frac{|set_\top|}{Area(r)} \geq kbound$ **then**
15: $\quad\quad$ enheap($H, \langle r, OE\_IBR, \frac{|set_\top|}{Area(r)}\rangle$)
16: **return** Search(H, h_Q)

After all unpruned regions have been pushed into the max-heap H (lines 12–15), the framework calls a search algorithm to find the top-k dense regions (line 16). We design two versions for the search. Both versions use the max-heap in the further processing of query regions, giving priority to the query regions with higher overestimated density values. For a region r and the set of objects $set_\top$ in r's indoor buffer region, Section 4.2 presents a one-pass search algorithm that checks all objects in $set_\top$ in a single pass to derive the object load for r, and Section 4.3 presents an improved version that searches in two passes and only derives the object loads when necessary.

Algorithm 3 overestimates (underestimates) a query region r's load based on its indoor buffer (core) region. It uses a depth-first search (line 2) via the R-tree for the indoor partitions. The load is overestimated as the count of all objects that are either in a leaf node fully in ibr (lines 7–9) or having their own reported locations in ibr (lines 12–15). Also, the load is underestimated as the count of all objects whose last reported location is in icr (lines 10–11 and 16–17). Algorithm 3 returns the set $set_\top$ of overestimated objects and the set $set_\perp$ of underestimated objects (line 3).

Algorithm 3 COUNT4ibcRs(Indoor buffer region ibr, Indoor core region icr, Partition R-tree R_P)

1: $set_\top \leftarrow \varnothing$; $set_\perp \leftarrow \varnothing$; $node \leftarrow R_P.root$
2: DFS($node$)
3: **return** $(set_\top, set_\perp)$
4: **function** DFS(Partition R-tree node $node$) ▷ Depth-first search
5: $\quad$ **if** $node$ is a leaf node **then**
6: $\quad\quad$ **for** each leaf entry le in $node$ **do**
7: $\quad\quad\quad$ **if** le is fully contained in ibr **then**
8: $\quad\quad\quad\quad$ **for** each object o in le **do**
9: $\quad\quad\quad\quad\quad$ add o to $set_\top$
10: $\quad\quad\quad\quad\quad$ **if** $o.loc$ falls in icr **then**
11: $\quad\quad\quad\quad\quad\quad$ add o to $set_\perp$
12: $\quad\quad\quad$ **else if** le overlaps ibr **then**
13: $\quad\quad\quad\quad$ **for** each object o in le **do**
14: $\quad\quad\quad\quad\quad$ **if** $o.loc$ falls in ibr **then**
15: $\quad\quad\quad\quad\quad\quad$ add o to $set_\top$
16: $\quad\quad\quad\quad\quad\quad$ **if** $o.loc$ falls in icr **then**
17: $\quad\quad\quad\quad\quad\quad\quad$ add o to $set_\perp$
18: $\quad$ **else**
19: $\quad\quad$ **for** each child node $child$ in $node$ **do**
20: $\quad\quad\quad$ **if** $child.mbr$ overlaps ibr **then**
21: $\quad\quad\quad\quad$ DFS($child$)

4.2 One-Pass Search

One-pass search is formalized in Algorithm 4. It keeps processing the query regions through the max-heap until the top-k regions are found. If the query region r deheaped from H is associated with a density overestimated by counting for its indoor buffer region (indicated by flag value OE_IBR on line 4), the search algorithm gets the overestimated object set $set_\top$ and underestimated object set $set_\perp$ from the hash table h_Q (line 5). Note that the involved objects in $set_\perp$ are certainly in r, and their counts are directly added to the object load $count$ (line 6). Afterwards, Algorithm 4 continues to update the object load inside region r by only going through the variable part of objects in $set_\top \setminus set_\perp$ (lines 6–11). Specifically, it calculates the count value according to Definition 1, by including the presences from all objects whose uncertainty region overlaps or fully contains region r. After the count value is obtained for the query region r, the algorithm enheaps region r to H with a flag value IR and r's density (line 12). Flag value IR indicates that the density is already computed for r. When such a query region r is deheaped in a future iteration (line 13), r is directly added into the result (line 14). If the result already contains k regions, the algorithm returns the result (line 15) as no other query regions can have higher density according to the second property in Pruning Rule 1.

Algorithm 4 Search1Pass(Max-heap H, Hash table h_Q)

1: $result \leftarrow \varnothing$
2: **while** H is not empty **do**
3: $\quad \langle r, flag, density\rangle \leftarrow$ deheap(H)
4: $\quad$ **if** $flag$ is OE_IBR **then** ▷ Overestimate for $\Theta^{\triangleright}_I(r)$
5: $\quad\quad (set_\top, set_\perp) \leftarrow h_Q[r]$
6: $\quad\quad count \leftarrow |set_\perp|$
7: $\quad\quad$ **for** each object $o \in set_\top \setminus set_\perp$ **do**
8: $\quad\quad\quad$ **if** $UR_I(o.loc)$ is fully contained in r **then**
9: $\quad\quad\quad\quad count \leftarrow count + 1$
10: $\quad\quad\quad$ **else if** $UR_I(o.loc) \cap r \neq \varnothing$ **then**
11: $\quad\quad\quad\quad count \leftarrow count + \phi^{\Gamma}_r(o)$
12: $\quad\quad$ enheap($H, \langle r, IR, \frac{count}{Area(r)}\rangle$)
13: $\quad$ **else** ▷ $flag$ is IR and the density is computed for r
14: $\quad\quad$ add r to $result$
15: $\quad\quad$ **if** $|result| = k$ **then return** $result$

For an encountered query region r, Algorithm 4 checks the variable part of objects in $set_\top \setminus set_\perp$ associated to r. It computes r's density (lines 4–12) directly in a single pass, which is time-consuming and may not pay off in case that r's density is not

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2018.2799215, IEEE Transactions on Knowledge and Data Engineering

sufficiently high to qualify it for the top-k result. Motivated as such, we proceed to design an improved search algorithm that computes density for region r in two passes and the second pass, which computes the concrete density, is evoked for r only when the first pass does not rule out r from the top-k result.

4.3 Improved Search

The improved search is formalized in Algorithm 5. For an encountered query region r with overestimated density based on its indoor buffer region (line 4), the improved search algorithm calls Algorithm 6 to get a tighter upper bound from the count of objects that may be in r at query time t_c (line 6).

We use function *OverCount*(r) to obtain the number of objects whose uncertainty region overlaps with a region r. The following lemma gives a tighter upper bound of r's indoor density.

Lemma 7 (OverCount Upper Bound).
$\tau_O(r) \leq \frac{OverCount(r)}{Area(r)} \leq \frac{COUNT(\Theta_I^{\triangleright}(r))}{Area(r)}$.

Proof 7. Based on Definitions 1 and 2, we have $\lambda_O(r) \leq OverCount(r)$. For any object o whose uncertainty region overlaps with r, given its last report time t_l, its shortest indoor distance to r must be shorter than $V_{max} \cdot (t_c - t_l)$. Due to Definition 5 and the fact that t_l is no older than the minimum time in $OIPT$, such an o must be inside $\Theta_I^{\triangleright}(r)$ at time t_l. As $\Theta_I^{\triangleright}(r)$ may contain other objects, we have $OverCount(r) \leq COUNT(\Theta_I^{\triangleright}(r))$. Thus, the lemma is proved.

Algorithm 5 SearchImproved(Max-heap H, Hash table h_Q)

1: $result \leftarrow \varnothing$
2: **while** H is not empty **do**
3: $\langle r, flag, density\rangle \leftarrow$ deheap(H)
4: **if** $flag$ is OE_IBR **then** ▷ Overestimate for $\Theta_I^{\triangleright}(r)$
5: $(set_\top, set_\bot) \leftarrow h_Q[r]$
6: $(count, set_u) \leftarrow$ OverCount$(r, set_\top, set_\bot, t_c)$
7: **if** $set_u = \varnothing$ **then**
8: enheap($H, \langle r, IR, \frac{count}{Area(r)}\rangle$)
9: **else**
10: $h_Q[r] \leftarrow (set_u, \varnothing)$
11: enheap($H, \langle r, OE\_IR, \frac{count}{Area(r)}\rangle$)
12: **else if** $flag$ is OE_IR **then** ▷ Overestimate for r
13: $(set_u, \varnothing) \leftarrow h_Q[r]$
14: $count \leftarrow density \cdot Area(r) - |set_u|$
15: $count \leftarrow count +$ COUNT$_u(r, set_u, t_c)$
16: enheap($H, \langle r, IR, \frac{count}{Area(r)}\rangle$)
17: **else** ▷ $flag$ is IR and the density is computed for r
18: add r to $result$
19: **if** $|result| = k$ **then return** $result$

In particular, Algorithm 6 works on the objects returned by the indoor buffer region based overestimation (Algorithm 3). It adds the count of objects in $set_\bot$ to the certain counting part (line 1) and processes each object in $set_\top \setminus set_\bot$ in a nested loop (lines 2–6). If an object o's uncertainty region $UR_I(o.loc)$ at query time t_c is fully contained in query region r, the counting is incremented by 1 (lines 3–4). If $UR_I(o.loc)$ only overlaps with r, o's is counted in the uncertain part of the counting and o is put into a set set_u that contains all objects that need further processing (lines 5–6). Last, the sum of the two counts and set_u are returned (line 7).

Algorithm 6 OverCount(Indoor region r, Object set $set_\top$, Object set $set_\bot$, Current time t_c)

1: $count_c \leftarrow |set_\bot|$; $count_u \leftarrow 0$; $set_u \leftarrow \varnothing$
2: **for** each object $o \in set_\top \setminus set_\bot$ **do**
3: **if** $UR_I(o.loc)$ is fully contained in r **then**
4: $count_c \leftarrow count_c + 1$
5: **else if** $UR_I(o.loc) \cap r \neq \varnothing$ **then**
6: $count_u \leftarrow count_u + 1$; add o to set_u
7: **return** $(count_c + count_u, set_u)$

Back in the improved search (Algorithm 5), if no uncertain object set is returned (line 7), query region r, a flag value IR, and r's density are pushed into the max-heap (line 8). The flag IR indicates that the density computation is finished for r. Otherwise, the uncertain object set set_u is added into the hash table (line 10), $\varnothing$ is included here in order to keep a uniform format in h_Q. After that, r is enheaped with a flag value OE_IR (line 11) to indicate that r's current enheaped density is an overestimate and it still has some uncertainty to process. When such a query region r is deheaped in a future iteration (line 12), Algorithm 5 gets the certain counting part in order to avoid duplicated counting (line 14), and calls Algorithm 7 to further process the uncertain objects.

Algorithm 7 handles those objects whose uncertainty regions only overlap a given query region r. For each such an object o, the object calculates how much o's uncertainty region $UR_I(o.loc)$ is inside the query region r (line 3). Finally, the sum of all such uncertain counts is returned (line 4).

Algorithm 7 COUNT$_u$(Indoor region r, Object set set, Current time t_c)

1: $count_u \leftarrow 0$
2: **for** each object $o \in set$ **do**
3: $count_u \leftarrow count_u + \phi_r^{\Gamma}(o)$
4: **return** $count_u$

Back in Algorithm 5, the two parts of counts are summed (line 15) and the query region r is enheaped to H with a flag value IR and its density (line 16). Flag IR indicates that the density is already computed for the indoor region r. When r is deheaped in a future iteration, the process (lines 17–19) is the same as the counterpart in Algorithm 4.

In summary, Algorithm 5 overestimates the object count for a query region r in two passes. The two passes call Algorithms 6 and 7 to count the objects whose uncertainty regions are contained by and overlap r, respectively. Between the two passes, a tighter overestimated density enabled by Lemma 7 is assigned to r (line 11 in Algorithm 5) that is between a coarser overestimate (enabled by Algorithm 3 for r's indoor buffer region) and r's final density. In this two-pass way, we expect to avoid part of the expensive counting for more uncertain objects as the tighter overestimated density may be lower than other query regions' final densities that are either already computed or to be computed soon.

4.4 Performance Gain by Improved Search

We use TopkIDRs1Pass and TopkIDRsImprd to denote the overall process that employs the one-pass search and the improved search, respectively. The crucial part of both processes is the density computation for indoor regions in the query set Q. For conciseness,

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2018.2799215, IEEE Transactions on Knowledge and Data Engineering

we assume that each query region incurs approximately the same computational cost, and we use C_p to denote such a unit cost in the search. We intend to capture the performance difference between these two processes.

Let $q = |Q|$. Suppose that TopkIDRs1Pass computes the density for x query regions in the order of $r_1, r_2, \ldots, r_x$ before it returns all the top-k dense regions. Note that $k \leq x \leq q$ and $\overline{x} = \frac{k+q}{2}$. After termination, TopkIDRs1Pass still keeps $x - k$ query regions with flag IR in its max-heap. Likewise, TopkIDRsImprd computes or overestimates the density also for x query regions (although in a different order perhaps) before it returns the top-k results. After its termination, TopkIDRsImprd keeps $x-k$ query regions in its max-heap but each of these regions is with a flag IR or OE_IR. The number of regions with either flag is expected to be $\frac{x-k}{2}$.

For a query region, the cost of overestimating its density is significantly lower than that of computing its concrete density. The performance gain by TopkIDRsImprd is expected to be $\frac{x-k}{2} \cdot C_p = \frac{(k+q)/2-k}{2} \cdot C_p = \frac{q-k}{4} \cdot C_p$. The performance gain increases when the query region set Q is larger and decreases when more regions are to be returned. When k equals q, i.e., we need to compute the concrete density for the entire Q, both processes perform the same. The brief analysis is consistent with the experimental results to be shown in Section 5.1.2.

5 Experimental Studies

All algorithms are implemented in Java, and run on a Windows 10 Pro PC with a 3.10GHz Core i3 CPU. We evaluate our proposals using synthetic and real data sets.

5.1 Experiments on Synthetic Data

5.1.1 Settings

Indoor Space. We use a real floor plan of a shopping mall, each floor taking 60m $\times$ 60m with 100 rooms and 4 staircases.[6] The irregular indoor partitions are decomposed into smaller but regular ones, yielding 141 partitions and 220 doors. We duplicate the original floor plan ten times to generate a larger floor plan with 1410 partitions and 2200 doors in total. The original stairways are used as doors to connect the ten duplicates. All the partitions are indexed by an R-tree for its high flexibility, compared to alternatives like grids or other spatial indexes, at indexing arbitrary shapes and layouts of indoor partitions. To speed up the search, the 10 duplicates are indexed by 10 child node entries at the root level. The tree fan-out is set to 20 on all other levels. The whole tree is approximately 7MB and it is kept in memory.

To represent the indoor topology, we use the accessibility graph as well as the *P2D* and *D2P* mappings from a previous work [23]. These structures allow us to access doors and partitions quickly by their IDs. Specifically, $D2P(d_k)$ maps a door d_k to a pair of partitions p_i and p_j such that an object can move from partition p_i to p_j (or in the opposite direction) through door d_k. Conversely, $P2D(p_k)$ maps a partition p_k to all the doors through which an object can enter or leave p_k. In addition, the door-to-door distance matrix [23] is pre-computed to speed up distance-related operations in our search, e.g., the determination of indoor buffer and core regions.

Indoor Moving Objects. We generate moving objects in the large floor plan for a lifespan of two hours using the indoor mobility data generator Vita [21]. Initially, we distribute 5K to 20K objects evenly into the entire indoor space. Gradually, new objects are added to enter the indoor space. The number of entering objects at every ten seconds follows the Poisson distribution with mean $\lambda = 1$. Object maximum speed constraint is set to $V_{max} = 1m/s$. An object's movement follows the random waypoint mobility model [17]. For the entire two-hour simulation, we record objects' exact locations every second and store them in spatiotemporal trajectories for objects. These trajectories with exact locations and times form the ground truth in our experiments.

OIPT. The $OIPT$ in our experiments is maintained according to the ground truth as follows. Each object sends updates to $OIPT$ at varying frequencies. In particular, an object keeps silent for 1 to 20 seconds after it has sent the latest update. An update from object o consists of a time stamp t and an exact location l, where l is a random location within the sampling range of 0.5 meter from o's exact location at time t. When an update is received by the $OIPT$, the old record for the object will be replaced by the new information.

Queries. The indoor query regions (Q) in a top-k dense region query are categorized into three groups, as shown in Table 4. When issuing a query, certain numbers of the three types of query regions are generated respectively. Tarjan's algorithm [12] is used to make sure all portions of a ir_3-type query region are connected. Each query with a fixed set of Q is processed at 10 random online time stamps within the data set lifespan, i.e., against 10 instances of $OIPT$. The average performance results are reported in the paper.

Table 4
Types of Indoor Query Regions in Q

Type	Meaning
ir_1	An incomplete indoor partition
ir_2	A complete indoor partition
ir_3	A combination of two or more ir_1 regions, of two or more ir_2 regions, or of both types of regions. It should be a self-connected part.

Other Settings. We also vary other parameters in the experiments, namely k, $|O|$, $|Q|$ and Δt. As the difference between the current (query) time and the minimum time in $OIPT$, $\Delta t = t_c - t_{min}$ indicates how old the data is for a search request. We also compare the different distance decaying functions (see Section 3.3) used in computing object presence and indoor density. Table 5 lists the parameter settings with default values in bold.

Table 5
Parameter Settings on Synthetic Data

Parameters	Settings
k	1, 3, 5, . . . , **15**
$\|O\|$	5K, **10K**, 15K, 20K
$\|Q\|$ (% of total indoor partitions)	2%, 4%, 6%, . . . , **10%**, . . . , 14%
Fractions of ir_1, ir_2, ir_3 in Q	40% , 50%, 10%
Δt (s)	1, 2, 3, 4, **5**, . . . , 10
Distance Decaying Function	CL, LDL, I1PL, I2PL, **EDL**

5.1.2 Top-k Search Efficiency

First, we compare the two top-k search methods, namely TopkIDRs1Pass and TopkIDRsImprd (see Section 4.4) in terms of efficiency. Either process runs 20 times for each particular single setting. We measure the average execution time and the *pruning ratio*. The latter is the fraction of moving objects excluded from the expensive computation of object presence.

As no existing work solves the defined problem in indoor contexts, we develop five alternatives to compare with our methods. A

6. http://goo.gl/iRbM0Q

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2018.2799215, IEEE Transactions on Knowledge and Data Engineering

natural baseline computes the load for each query region $r \in Q$ by directly counting the total number of objects whose last reported location in $OIPT$ is contained by r. The top-k query regions with the highest densities are then returned by a full ranking. This method ignores the location uncertainty in the positioning data, and we call it directly counting (**DC**). There also exist some variant algorithms that employ our indoor density definition. First, a region-oriented nested-loop method (**NLRegion**) computes the density for each query region $r \in Q$ by summing up all moving objects' contributions (presences) to r's density, and returns the top-k regions. The crucial part of NLRegion is that the algorithm should iterate through all the objects when computing each region's density. NLRegion's major time cost is estimated as $q \cdot C_{pRegion}$ where $q = |Q|$ and $C_{pRegion} \gg C_p$, the unit cost of our methods to compute r's density (see Section 4.4). Next, we develop an object-oriented nested-loop method (**NLObject**) that exchanges the orders of the outer loop and the inner loop in NLRegion. In particular, NLObject iterates through each moving object contained in $OIPT$, and calculates its presence in the query regions from Q. The top-k results are returned after the object presences are summed up for each query region. NLObject does not determine an object's uncertainty region repeatedly when iterating on query regions and thus is more efficient than NLRegion. Furthermore, two improved versions of NLRegion reduce the search space by pruning the objects outside r's buffer region. Specifically, **NLwgbr** uses general buffer regions (simplified to minimum bounding rectangles for fast pruning) to prune objects with zero contribution whereas **NLwibr** uses indoor buffer regions. Their time costs are estimated as $q \cdot C_{pNLgbr}$ and $q \cdot C_{pNLibr}$, respectively, where $C_{pNLgbr} > C_{pNLibr} > C_p$. The cost of our methods is upper bounded by $x \cdot C_p$, where x ($k \leq x \leq q$) is the number of query regions for which a search overestimates or computes the density.

We use random queries with the default settings. The results of efficiency comparison on all implemented methods are reported in Table 6. Clearly, DC has the lowest execution time and memory cost since it immediately uses the location reports in $OIPT$ to compute the densities and omits the computations on uncertainty regions. However, DC's effectiveness is very poor, as to be presented in Sections 5.1.3 and 5.2. In other words, the search results returned by DC are of very low quality compared to other methods and the ground truth.

Different from DC, the other methods all consider the location uncertainty. We call them uncertainty model (**UM**) based search methods. Among them, our proposed methods significantly outperform all nested-loop alternatives. Compared to our methods, NLRegion is slower by orders of magnitude, its two improved versions and NLObject are several times slower. The pruning ratio explains the reason behind—the bounds of indoor density in our methods are very effective in pruning objects. Also, TopkIDRsImprd has a higher pruning capability than TopkIDRs1Pass, which is consistent with the analysis in Section 4.4. On the other hand, our proposed methods also have lower memory consumptions compared to their alternatives. It is noteworthy that TopkIDRsImprd uses more memories than TopkIDRs1Pass since it produces more intermediate data. Moreover, NLwibr requires more memories than NLwgbr as it has to keep the indoor buffer regions in the memory.

In the sequel, we omit DC and only compare its effectiveness with UM search methods in Section 5.1.3. We also omit the inefficient UM search methods and focus on our proposed search

Table 6
Efficiency Comparison in Default Setting

Algorithms	Running time (millisec.)	Pruning ratio (%)	Memory used (MB.)
TopKIDRs1Pass	399.7	81.56	147.8
TopKIDRsImprvd	365.7	**85.02**	156.1
DC	**68.5**	-	**2.2**
NLRegion	148386.2	0	342.5
NLObject	2248.1	0	321.3
NLwibr	1082.2	60.74	68.6
NLwgbr	1597.7	34.85	51.2

methods, i.e., TopkIDRs1Pass and TopkIDRsImprd. We break down TopkIDRs1Pass's cost into 2 parts: counting object number based on the indoor buffer and core regions (COUNT4ibcRs), and density computation in one pass (COUNT1Pass). Likewise, we break down TopkIDRsImprd's cost into 3 parts: COUNT4ibcRs, counting objects whose uncertainty region is contained in a query region r and identifying those objects whose uncertainty region only overlaps with r (OverCount), and computing the latter set of objects' presence in r (COUNT$_u$). In the implementation, the max heap is constructed on the fly for an issued query. We include the cost of heap construction in the part of COUNT4ibrRs.

Next, we investigate the effect of k by varying it from 1 to 15. The results are reported in Figure 5. Overall, increasing k makes the execution time grow moderately for both search methods but never higher than 400 milliseconds. The performance is thus very competent for online search. In both methods, COUNT4ibcRs is insensitive to k and costs only a small part of the total execution time. In contrast, the subsequent COUNT1Pass and OverCount+COUNT$_u$ run longer as k is larger. A larger k involves more query regions in these phases that compute, rather than overestimate, the indoor densities. TopkIDRsImprd always outperforms TopkIDRs1Pass; the former makes relatively fine overestimates after COUNT4ibcRs in two passes, whereas the latter directly computes the densities in one pass. The performance gain shrinks for larger ks, which is again consistent with the analysis in Section 4.4.

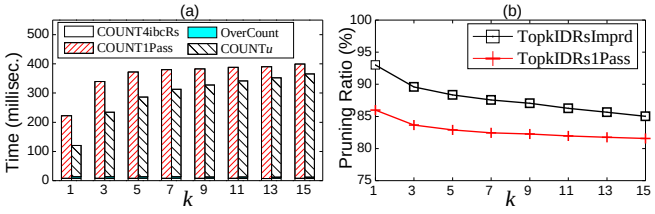


Figure 5. Efficiency vs. k

Further, we fix $k = 15$ and vary $|O|$ from 5K to 20K. The results are reported in Figure 6. More moving objects result in longer execution time for both search methods. The execution time of COUNT4ibcRs also increases when O is larger, since

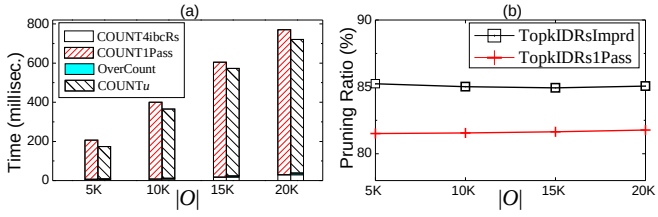


Figure 6. Efficiency vs. $|O|$

more objects can be in a fixed indoor buffer (core) region. More objects also cause the subsequent phases to finish in more time,

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2018.2799215, IEEE Transactions on Knowledge and Data Engineering

as the indoor region density computations involve more objects. Nevertheless, both methods can still return the top-15 results in less than 771 milliseconds even $|O|$ is up to 20K, totally acceptable for online search.

Moreover, we investigate the query region set Q's effect. The size of Q is set as varying percents of the total 1410 indoor partitions. We first use random compositions of the three region types (Table 4) in each Q, and the results are shown in Figure 7. The overall execution time of both methods increases markedly when $|Q|$ increases. For every phase of the search, the time cost grows stably since more query regions from Q are processed. Both methods need to determine the indoor buffer and core regions for each query region r in Q, and thus the cost of COUNT4ibcRs grows steadily when Q is larger. Meanwhile, the phases to compute concrete indoor densities need to process more query regions and thus their costs also increase. Also, TopkIDRsImprd outperforms TopkIDRs1Pass more visibly for larger Qs. The increasing performance gain is again consistent with the analysis in Section 4.4.

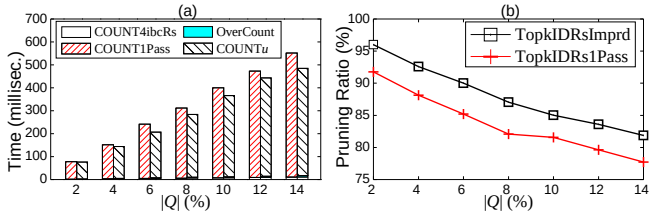


Figure 7. Efficiency vs. $|Q|$

In order to study the effect of query region types, we use homogeneous query regions in Qs. The results are shown in Figure 8. The cost is the lowest for ir_1-type since this type yields smaller indoor buffer (core) regions, whereas ir_3-type incurs the highest cost since this type complicates the determination of indoor buffer (core) regions by involving more indoor partitions and doors. The cost for ir_2-type is in-between since this type's complexity is between the other two.

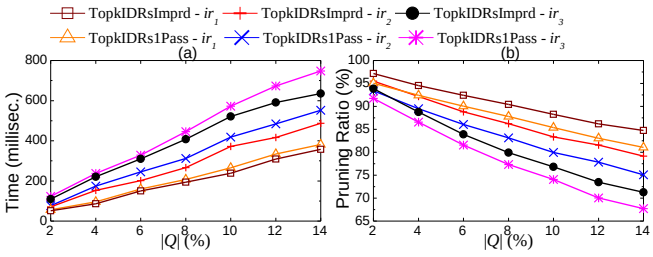


Figure 8. Efficiency vs. Types in Q

We further study the effect of Δt. We choose particular time stamps to issue the search such that Δt varies from 1s to 10s. Referring to the results in Figure 9, both methods incur significantly increased execution time for larger Δt. A larger Δt means not only a larger indoor buffer region for a query region r, but also a larger uncertainty region $UR_I(loc)$ for an indoor object o. As Δt increases, the execution time of COUNT4ibcRs and OverCount grows stably as they are more affected by the number of objects in the query regions' indoor buffer regions. In contrast, the cost of COUNT$_u$ and COUNT1Pass increases more rapidly since they are affected by the objects in indoor buffer region as well as the enlargement of objects' uncertainty regions. Note that the variable part of objects involved in OverCount and COUNT1Pass also become more as r's indoor core region becomes smaller and the density lower bound approaches to zero when Δt increases. Nevertheless, both methods can still return the top-15 results within 920 milliseconds. This shows that our methods are efficient for online search even when $OIPT$ contains relatively old positioning information.

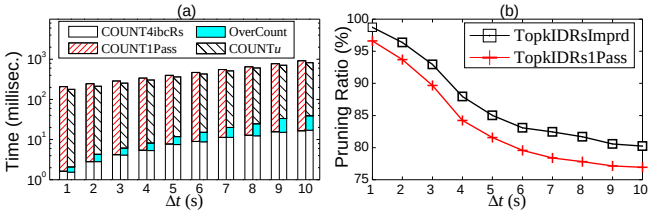


Figure 9. Efficiency vs. Δt

Last, we study the effect of DDFs. The results are shown in Figure 10. Clearly, TopkIDRsImprd outperforms TopkIDRs1Pass in all tested cases. The overall execution time increases when the DDF becomes more complex because a more complex DDF makes it slower to compute the concrete density for query regions. Nevertheless, a search with any of the five DDFs can return the top-k results in less than 400 milliseconds. Taking a deeper look, the execution time of COUNT4ibcRs and OverCount stays stable for different DDFs. A DDF does not affect the estimation of indoor density; neither does it affect the object pruning.

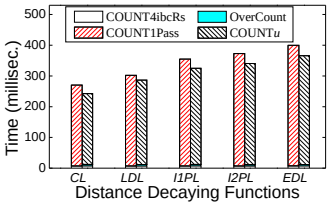


Figure 10. Efficiency vs. DDFs

5.1.3 Top-k Result Effectiveness

We also study the top-k search effectiveness with respect to the ground truth that is calculated according to the moving objects' true locations in the trajectories described in Section 5.1.1. As the UM search methods return the same result, we run TopkIDRsImprd at 10 random t_cs and compare it with DC in the same settings. To lower the biases caused by positioning reports, for each query time t_c, we generate 20 $OIPT$ instances by modifying the positioning reports of the moving objects that update $OIPT$. We run TopkIDRsImprd (UM) and DC 20 times at each of the 10 t_cs and report their average effectiveness measures.

We consider two metrics. *Recall* measures the ratio of true top-k dense regions in the returned top-k results. It is equal to precision in the top-k search. *Kendall coefficient* τ is a measure of rank correlation. In our setting, it captures the similarity between the ranking of top-k search result (φ_r) and that of top-k ground truth dense regions (φ_g). Let cp be the number of dense region pairs (r_i, r_j) whose rankings in φ_r and φ_g are concordant, i.e., r_i is before (after or in tie with) r_j in both rankings. Let dp be the number of dense region pairs (r_i, r_j) whose rankings in φ_r and φ_g are discordant. The Kendall coefficient is $\tau = \frac{cp-dp}{0.5k(k-1)}$. If the agreement between the two rankings is perfect (i.e., they are identical), τ equals 1. In contrast, τ is -1 if one ranking is the reverse of the other. When φ_r and φ_g do not cover the same set of objects, we make slight changes to them in order to compare them. Suppose $k = 3$, φ_r is $\langle A, B, C\rangle$ and φ_g is $\langle B, D, E\rangle$. In order to measure the Kendall coefficient between the two rankings, we extend φ_r to $\langle A, B, C, D, E\rangle$ and φ_g to $\langle B, D, E, A, C\rangle$. The elements we add into either ranking have the same ordering value, e.g., elements D and E are ranked 4th in the modified φ_r.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2018.2799215, IEEE Transactions on Knowledge and Data Engineering

First, we use default values but vary k from 3 to 15 and $|O|$ from 5K to 20K. The results of effectiveness are reported in Figure 11. Referring to Figure 11(a), the Kendall coefficient of both DC and UM increases moderately with an increasing k. Overall, DC's Kendall coefficient is lower than -0.13 in all the tested cases. In contrast, the Kendall coefficient of our search method is always positive and above 0.68 when $k \geq 9$, implying the high consistency between the ground truth and the results returned by our search. On the other hand, the cardinality of indoor moving objects only results in fluctuations of the both methods' Kendall coefficient due to data randomness. In this regard, our top-k search approach is not sensitive to the object cardinality. According to Figure 11(b), the recall of our top-k search is higher than 0.86 for the most tested cases while DC's recall is always less than 0.31. Our search is able to find almost all the densest indoor regions in most cases. For a fixed $|Q|$, larger ks tend to include more ground truth dense regions in the top-k search results, and therefore the two effectiveness measures improve when k is increased in our experiments.

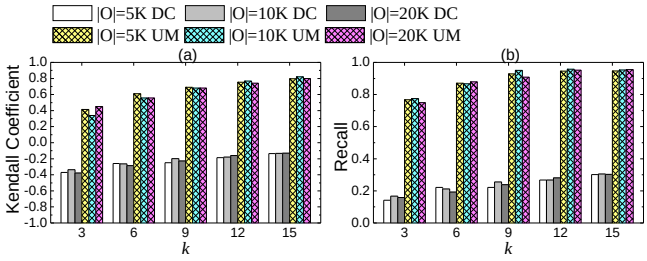


Figure 11. Effectiveness vs. k and $|O|$

Next, we vary Δt from 1s to 9s and test the effect of using different sizes of Q. The results are shown in Figure 12. Clearly, the Kendall coefficient and recall of UM decrease as $|Q|$ becomes larger. When more regions are in the query set Q, the search space becomes larger and more densities (and uncertainty regions) are to be computed and ranked. Consequently, the search results become less effective. The Kendall coefficient slightly decreases with decreasing Δt but it clearly stays above 0.72 in all cases, according to Figure 12(a). A larger Δt yields larger uncertainty regions for objects, which makes the ranking of uncertainty regions based on densities tend to be less close to the ground truth. The recall overall decreases as Δt increases, but it is always higher than 0.91, according to Figure 12(b). Thus, our online top-k search is very effective in finding the correct results even when the $OIPT$ contains old positioning information that is not updated. On the other hand, DC's both measures are very low since it ignores the location uncertainty.

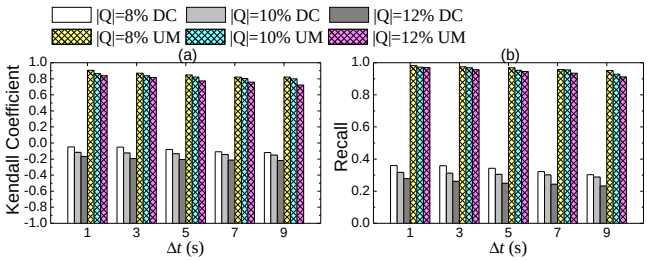


Figure 12. Effectiveness vs. Δt and $|Q|$

We further investigate the effect of query region types when varying Δt from 1s to 9s. Referring to Figure 13, both effectiveness measures of UM significantly beat those of DC and are very high for all tested cases. In particular, UM's Kendall coefficient is always above 0.79, and its recall is always above 0.92. Therefore, our search is very effective regardless of query region types. Interestingly, UM's Kendall coefficient when processing ir_3-type query regions is slightly higher than that for the other types. This indicates that our search is very stable to the changing composition of query regions. On the other hand, it can be seen that increasing Δt only slightly deteriorates the search effectiveness of our proposed method, but almost has no impact on the effectiveness of DC in the tested cases.

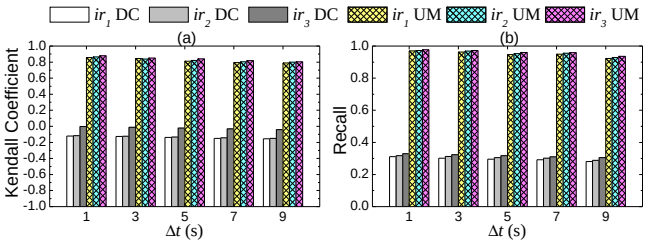


Figure 13. Effectiveness vs. Δt and Types of Q

Last, we investigate the search effectiveness using all default settings but different DDFs in TopkIDRsImprd. The results are reported in Figure 14. On the one hand, CL only gets a Kendall coefficient of 0.22 since the assumption of a uniform $UR_I(loc)$ reflects poorly the uncertainty of object movement in $UR_I(loc)$. In contrast, DDFs LDL, I1PL and I2PL achieve a Kendall coefficient of 0.59, 0.79 and 0.58 respectively. The default DDF EDL performs the best, which achieves a Kendall coefficient of 0.82. On the other hand, EDL is still the best in terms of the recall. These results show that, by considering the distance decaying effect in $UR_I(loc)$, the effectiveness of our top-k search improves significantly. In particular, it clearly pays off to introduce the complex EDL in computing object presence and indoor density.

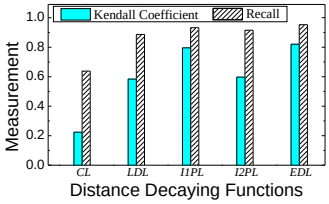


Figure 14. Effectiveness vs. DDF

5.2 Experiments on Real Data

In this part, we evaluate the top-k result effectiveness using real data collected in a university building. The indoor space for the experiments is a floor of 18m $\times$ 24m with 14 partitions and 16 doors. A Wi-Fi based indoor positioning system is deployed, which offers a positioning accuracy around 4 meters. We collect ground truth trajectories as well as the Wi-Fi positioning data for 35 persons who move according to their daily behaviors. Due to the actual floor use, we use $ir2$-type for all query regions in Q in the experiments. As the number of partitions is small, we set $k = 5$, and vary Δt from 3s to 8s and $|Q|$ from 40% (6 query regions) to 100% (14 query regions).

The effect of changing $|Q|$ and Δt on the effectiveness of the top-k search is shown in Figure 15. Clearly, our proposed UM method outperforms DC significantly in both measures of all the tested cases. An interesting case happens when $|Q| = 40\%$. As five of the six query regions are returned to form the search result, DC's Kendall coefficient and recall are both very high. However, the two measures decrease rapidly when we increase $|Q|$. Take a close look at UM, its Kendall coefficient in Figure 15(a) decreases with larger $|Q|$ as more query regions are involved in the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2018.2799215, IEEE Transactions on Knowledge and Data Engineering

search. Still, the Kendall coefficient is very close to 0.7 when all partitions are used as query regions. On the other hand, the Kendall coefficient decreases with larger Δt values. When Δt increases, an object's uncertainty region becomes larger, which reduces the accuracy of indoor densities computed in the top-k search. Thus, the search effectiveness tends to decrease. UM's recall is reported in Figure 15(b). Again, larger $|Q|$s involve more query regions in the search, and larger Δts lead to larger uncertain regions. Both factors render it more difficult to find the top-k dense regions, and thus the recall decreases. Nevertheless, UM's recall is still around 0.78 even all of the partitions are used as the query regions.

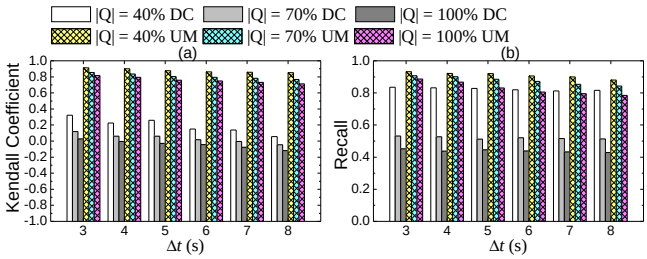


Figure 15. Effectiveness vs. Δt and $|Q|$ on University Data

Next, we fixed $k = 5$, $|Q| = 40\%$ and $\Delta t = 5s$, and test the effect of using different DDFs. The results are reported in Figure 16. Apparently, applying CL results in very poor search effectiveness since assuming uniform uncertainty regions is oversimplified. Interestingly, I2PL leads to a comparable Kendall coefficient and even slightly better recall in comparison with EDL. The university environment we use is relatively small and features many obstacles, creating many fixed indoor paths with different short lengths. This case renders I2PL very good at capturing the distance decaying effect in the movements of persons collecting the data, but leaves less room for more complex DDFs to play a role. Nevertheless, the default EDL is overall the best in that it helps achieve excellent effectiveness.

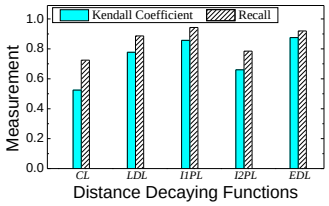


Figure 16. Effectiveness vs. DDF on University Data

6 Related Work

Querying Indoor Space Moving Objects. Indoor spaces are modeled [16], [19], [23], [28], [29] differently than outdoor spaces. Also, indoor positioning data is different than outdoor GPS data. Therefore, managing and querying indoor moving objects call for novel techniques.

Yang *et al.* [32] study continuous range monitoring over indoor moving objects tracked by RFID-like indoor positioning technologies. In particular, moving object positions are reported as the detection ranges of relevant positioning devices (e.g., RFID readers) that detect the objects. In the same setting, the authors study k nearest neighbor queries on indoor moving objects [33]. Note that both works [32], [33] return uncertain query results as object locations are unknown when they are not in any RFID's detection range. In order to improve the query result quality, Yu *et al.* [35] propose a particle filter based method to infer the undetected locations of RFID-tracked indoor moving objects. Unlike these studies, our work on indoor densities does not assume uniform distributions of locations in an object's uncertainty region.

Xie *et al.* [30], [31] propose techniques for processing indoor distance-aware spatial queries and joins over indoor moving objects whose positions are reported as probabilistic samples. The indoor location uncertainty handling techniques in [30], [31] focus on capturing and computing the uncertain indoor distances between query locations and moving objects. Working for the data format and problems different from those in this paper, the techniques in [30], [31] are not applicable to the search of indoor dense regions.

Ahmed *et al.* [2], [3] study offline indoor density queries by searching historical RFID indoor tracking data. Our work is different in several aspects. First, our work is not specialized only for RFID indoor tracking data. Second, our work addresses online search using the current data. Third, our work defines density with respect to the uncertainties in the positioning data whereas works [2], [3] consider no uncertainty when defining the density. Fourth, our work finds top-k dense geometric indoor regions but works [2], [3] return indoor semantic locations with densities higher than a threshold. Also using historical indoor RFID data, a recent work [24] finds the most frequented indoor POIs in the past. However, the techniques in work [24] are for historical data and cannot be applied to the online search studied in this paper.

Density Queries in Outdoor Settings. Assuming Euclidean spaces, Tao *et al.* [27] propose techniques to count spatio-temporal objects within a given spatial window during a given historical time interval. The proposed techniques are inapplicable to the indoor dense region problem in this paper because they do not support indoor regions and topology. Li *et al.* [22] propose techniques to cluster linearly moving objects in outdoor spaces. The proposed techniques do not apply to our setting where indoor object movements cannot be captured by linear models. Yiu and Mamoulis [34] propose density based and hierarchical methods to partition and cluster static objects on a spatial network. Their proposal does not support indoor regions and topology, and therefore does not solve our research problem.

Hadjieleftheriou *et al.* [8] formulate snapshot and period threshold density queries over outdoor objects moving according to known linear functions. Jensen *et al.* [15] study snapshot dense region queries in a Euclidean plane where objects move linearly. In order to improve the query result quality, Ni and Ravishankar [25] redefine the density and use small square neighborhoods to approximate arbitrary outdoor regions. In a similar setting, Hao *et al.* [9] study continuous density queries. These proposals [8], [9], [15], [25] are unsuitable for our problem because indoor objects in our setting move in unknown manners other than linear functions and the indoor topology is more complicated than the outdoor counterpart.

Huang and Lu [13] study online density region queries on moving objects in Euclidean spaces whose positions are approximated as sensor detection ranges. In the context of road networks, Li *et al.* [20] work on finding traffic density-based hot routes from historical trajectories, and Lai *et al.* [18] study continuous dense segment monitoring where object positions are described as offsets to their road segment ends. In contrast, the indoor moving object positions are captured as discrete indoor locations in our research.

7 Conclusion and Future Work

This research tackles the problem of finding the current top-k indoor dense regions from a set of user-defined query regions. Our approach uses online indoor positioning data that contains a latest indoor location report for each moving object. We design

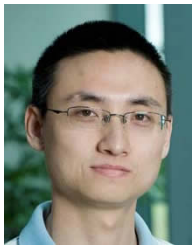
an appropriate indoor density definition amenable to the object location uncertainty caused by the discrete indoor positioning. We thoroughly analyze the data uncertainties with respect to the top-k search, and derive upper and lower bounds of indoor densities. The analysis outcomes enable us to devise efficient search algorithms. We conduct extensive experimental studies on synthetic and real data. The results demonstrate that our search algorithms are efficient, scalable, and effective. The top-k indoor dense regions returned by our search are highly consistent with ground truth, although we use online data only and assume no extra knowledge about the objects.

For future work, it is interesting to study continuous indoor density queries and indoor object clustering by applying the techniques proposed in this paper. Also, it may be possible to improve indoor density computing by learning indoor movement models from historical data. Moreover, it may make sense to compare our approach with alternatives for densities, e.g., computer vision and crowdsourcing, through interdisciplinary efforts with extra investment on devices and crowds.

REFERENCES

[1] InLocation Alliance. http://www.in-location-alliance.com/, 2016.
[2] T. Ahmed, T. B. Pedersen, and H. Lu. Finding dense locations in indoor tracking data. In *MDM*, pp. 189–194, 2014.
[3] T. Ahmed, T. B. Pedersen, and H. Lu. Finding dense locations in symbolic indoor tracking data: modeling, indexing, and processing. *GeoInformatica*, 21(1):119–150, 2017.
[4] T. Becker, C. Nagel, and T. H. Kolbe. A multilayered space-event model for navigation in indoor spaces. *3D Geo-Info*, pp. 61–77, 2008.
[5] M. D. Bjelland, et al. Human Geography: Landscapes of Human Activities. *McGraw-Hill Education*, 12th edition, 2013.
[6] C. Böhm. A cost model for query processing in high dimensional data spaces. *TODS*, 25(2):129–178, 2000.
[7] G. M. Giaglis and G. Lekakos. A taxonomy of indoor and outdoor positioning techniques for mobile location services. *SIGecom Exchanges*, pp. 19–27, 2003.
[8] M. Hadjieleftheriou, et al. On-line discovery of dense areas in spatio-temporal databases. In *SSTD*, pp. 306–324, 2003.
[9] X. Hao, X. Meng, and J. Xu. Continuous density queries for moving objects. In *MobiDE*, pp. 1–7, 2008.
[10] J. Hightower and G. Borriello. Location Systems for Ubiquitous Computing. *Computer*, 34:57–66, 2001.
[11] V. Honkavirta, et al. A comparative survey of WLAN location fingerprinting methods. In *WPNC*, pp. 243–251, 2009.
[12] J. Hopcroft and R. Tarjan. Algorithm 447: Efficient algorithms for graph manipulation. *Commun. ACM*, 16(6):372–378, 1973.
[13] X. Huang and H. Lu. Snapshot density queries on location sensors. In *MobiDE*, pp. 75–78, 2007.
[14] P. Jenkins, et al. Activity patterns of californians: Use of and proximity to indoor pollutant sources. *Atmospheric Environment*, 26A(12), 1992.
[15] C. S. Jensen, D. Lin, B. C. Ooi, and R. Zhang. Effective density queries on continuously-moving objects. In *ICDE*, page 71, 2006.
[16] C. S. Jensen, H. Lu, and B. Yang. Graph model based indoor tracking. In *MDM*, pp. 122–131, 2009.
[17] D. B. Johnson and D. A. Maltz. Dynamic source routing in ad hoc wireless networks. In *Mobile Computing*, pp. 153–181, 1996.
[18] C. Lai, L. Wang, J. Chen, X. Meng, and K. Zeitouni. Effective density queries for moving objects in road networks. In *WAIM*, pp. 200–211, 2007.
[19] J. Lee. A spatial access-oriented implementation of a 3-D GIS topological data model for urban entities. *GeoInformatica*, 8(3):237–264, 2004.
[20] X. Li, et al. Traffic density-based discovery of hot routes in road networks. In *SSTD*, pp. 441–459, 2007.
[21] H. Li, et al. Vita: A versatile toolkit for generating indoor mobility data for real-world buildings. *PVLDB*, 9(13):1453–1456, 2016.
[22] Y. Li, J. Han, and J. Yang. Clustering moving objects. In *SIGKDD*, pp. 617–622, 2004.
[23] H. Lu, X. Cao, and C. S. Jensen. A foundation for efficient indoor distance-aware query processing. In *ICDE*, pp. 438–449, 2012.
[24] H. Lu, et al. Finding Frequently Visited Indoor POIs Using Symbolic Indoor Tracking Data. In *EDBT*, pp. 449–460, 2016.
[25] J. Ni and C. V. Ravishankar. Pointwise-dense region queries in spatio-temporal databases. In *ICDE*, pp. 1066–1075, 2007.
[26] W. R. Ott. Human activity patterns: a review of the literature for estimating time spent indoors, outdoors, and in transit. In *Research Planning Conference on Human Activity Patterns*, pp. 1–38, 1989.
[27] Y. Tao, G. Kollios, J. Considine, F. Li, and D. Papadias. Spatio-temporal aggregation using sketches. In *ICDE*, pp. 214–225, 2004.
[28] E. Whiting, J. Battat, and S. Teller. Topology of Urban Environments. *CAADFutures*, pp. 114–128, 2007.
[29] M. F. Worboys. Modeling indoor space. In *ISA*, pp. 1–6, 2011.
[30] X. Xie, H. Lu, and T. B. Pedersen. Efficient distance-aware query evaluation on indoor moving objects. In *ICDE*, pp. 434–445, 2013.
[31] X. Xie, H. Lu, and T. B. Pedersen. Distance-aware join for indoor moving objects. *TKDE*, 27(2):428–442, 2015.
[32] B. Yang, H. Lu, and C. S. Jensen. Scalable continuous range monitoring of moving objects in symbolic indoor space. In *CIKM*, pp. 671–680, 2009.
[33] B. Yang, H. Lu, and C. S. Jensen. Probabilistic threshold k nearest neighbor queries over moving objects in symbolic indoor space. In *EDBT*, pp. 335–346, 2010.
[34] M. L. Yiu and N. Mamoulis. Clustering objects on a spatial network. In *SIGMOD*, pp. 443–454, 2004.
[35] J. Yu, W. Ku, M. Sun, and H. Lu. An RFID and particle filter-based indoor spatial query evaluation system. In *EDBT*, pp. 263–274, 2013.

Huan Li is currently working toward the PhD degree in the College of Computer Science, Zhejiang University, China. He was a research assistant in the Department of Computer Science, Aalborg University in the year 2014. His research interests include mobile/pervasive computing and spatial-temporal data management. He has published two relevant papers in VLDB and UbiComp.



Hua Lu is an associate professor in the Department of Computer Science, Aalborg University, Denmark. He received the BSc and MSc degrees from Peking University, China, and the PhD degree in computer science from National University of Singapore. His research interests include database and data management, geographic information systems, and mobile computing. He has served as PC cochair or vice chair for ISA 2011, MUE 2011 and MDM 2012, demo chair for SSDBM 2014, and PhD forum cochair for MDM 2016. He is a senior member of the IEEE.

Lidan Shou received the PhD degree in computer science from the National University of Singapore. He is a professor with the College of Computer Science, Zhejiang University, China. His research interests include spatial databases, data access methods, visual and multimedia databases, and web data mining.

Gang Chen received the PhD degree in computer science from Zhejiang University. He is a professor in the College of Computer Science at Zhejiang University. He has successfully led the investigation in research projects which aim at building China's indigenous database management systems. His research interests range from relational database systems to large-scale data management technologies.

Ke Chen received the PhD degree in computer science from Zhejiang University. She is an associate professor in the College of Computer Science, Zhejiang University. Her research interests include spatial-temporal data management, web data mining, and data privacy protection.