Continuous Social Distance Monitoring in Indoor Space

Harry Kai-Ho Chan
Roskilde University, Denmark
kai-ho@ruc.dk

Huan Li
Aalborg University, Denmark
lihuan@cs.aau.dk

Xiao Li
Roskilde University, Denmark
xiaol@ruc.dk

Hua Lu
Roskilde University, Denmark
luhua@ruc.dk

ABSTRACT

The COVID-19 pandemic has caused over 6 million deaths since 2020. To contain the spread of the virus, social distancing is one of the most simple yet effective approaches. Motivated by this, in this paper we study the problem of continuous social distance monitoring (SDM) in indoor space, in which we can monitor and predict the pairwise distances between moving objects (people) in a building in real time. SDM can also serve as the fundamental service for downstream applications, e.g., a mobile alert application that prevents its users from potential close contact with others. To facilitate the monitoring process, we propose a framework that takes the current and future uncertain locations of the objects into account, and finds the object pairs that are close to each other in a near future. We develop efficient algorithms to update the result when object locations update. We carry out experiments on both real and synthetic datasets. The results verify the efficiency and effectiveness of our proposed framework and algorithms.

PVLDB Reference Format:
Harry Kai-Ho Chan, Huan Li, Xiao Li, and Hua Lu. Continuous Social Distance Monitoring in Indoor Space. PVLDB, 15(7): 1390-1402, 2022.
doi:10.14778/3523210.3523217

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/harryckh/CSDM-public.

1 INTRODUCTION

The COVID-19 pandemic has affected almost all countries since the beginning of 2020. As of March 8, 2022, there have been more than 446 million confirmed cases and caused more than 6 million deaths[1]. To prevent the virus spread, the World Health Organization has suggested many guidelines, such as keeping at least 1 meter between people (i.e., social distancing), wearing face masks, and maintaining good hygiene practices. Since the virus mainly spreads to people in close contact, social distancing is one of the effective approaches to lower the infection rate [2, 20].

On the other hand, people spend more time in various indoor venues (e.g., airports, malls and office buildings) than outdoors. Multiple studies [14, 18] disclose that people spend on average 87% of their time indoors. Motivated by this, in this paper we study the problem of continuous social distance monitoring (SDM) in an indoor space, in which we keep track of the pairwise distances between the people in a building in real time. This can help to reinforce the social distancing requirement in indoor spaces as an effective means to contain the spread of the virus. For example, a gallery or museum can provide a guide app that integrates with social distance monitoring. With such an app, the visitors can keep track of the distances with others. As another example, the staff in those high-risk workplaces (e.g., elderly care centers and quarantine hotels) could avoid contact with each other to lower the infection risk as much as possible. Moreover, it can also provide statistics from a global perspective, such as calculating the contact frequency and identifying hotspots that close contact is likely to happen. These can give insights to the policy-makers on the effectiveness of the social distancing, and help them to further refine the restrictions, e.g., limiting the number of people in certain areas at peak hours.

In addition, our problem can be applied in other settings. For example, monitoring the distances between the pick-and-put robots in a warehouse to avoid possible collisions. It can also serve as the foundation for further indoor data analysis and data-mining, such as online clustering and classification.

Nevertheless, if we only monitor the distances between the people at the current time, it does not help to achieve social distancing. Thus, we also have to *predict* the distances in advance. Specifically, given a set of moving objects in an indoor space, SDM identifies all object pairs that are going to form close contact, i.e., having a distance smaller than a pre-defined threshold ϵ (e.g., 1 meter) within a near future (e.g., in 5 seconds or so). Such monitoring and predictions can serve as the fundamental service for other downstream applications. For example, a preventive application can suggest further actions to those contact object pairs by giving them an alert. Or, a routing application can find alternative routes for users to avoid close contacts en route. Depending on the application need, the actual alert frequency should be adjustable to the users to avoid overwhelming alerts. For example, the frequency can be lower if the user is in a private room, whereas it should be higher if the user is in a public space, suspiciously infected, or a close contact of an infected person. As object locations in the future are uncertain by nature, the distances between pairs are probabilistic. In this paper, we calculate the expected distances based on their future possible locations, and report object pairs with that smaller than ϵ.

Moreover, unlike the accurate GPS positioning records outdoors, raw indoor positioning data is uncertain and discrete in nature due to the limited indoor positioning technologies [4]. For example, an

Proceedings of the VLDB Endowment, Vol. 15, No. 7 ISSN 2150-8097.
doi:10.14778/3523210.3523217

[1]https://covid19.who.int

indoor object's location is not always up-to-date as it might update only once in a few seconds. We need to take the uncertainty into consideration when computing distances between objects.

Yet another challenge is that we need to keep a balance between the frequency of objects' updates and the accuracy of the results. If the objects update their locations too frequently, it will bring about pressures to the network load, and consume more energy on the object's device (usually a smartphone). On the other hand, if the updates are too sparse, the accuracy of the computed distances will be low. Therefore, we need to design a mechanism that minimizes the number of location updates, while maintaining the accuracy of the predicted distances and thus keeping the quality of the results.

To address these challenges, we propose an integrated client-server framework to facilitate the whole monitoring process. In particular, given objects' last reporting locations only, the server computes the expected distance between the object pairs, from the current time to a (near) future time, as a sliding time window. Subsequently, the server maintains those object pairs that satisfy the distance threshold in the result. When an object's location updates or when a new object is inserted, the server updates the monitoring results accordingly in an incremental fashion.

While some existing works [40] target minimizing the communication cost between the server and clients (i.e., objects), our focus is to compute the result efficiently in the system. With the ubiquity of the high-speed 5G network, the communication delay between the server and objects will become less crucial, and thus in this paper we aim at optimizing the server's running time to achieve a real-time response. Nevertheless, our framework requests objects to update their locations only when necessary.

In this paper, we adopt the online indoor positioning data setting in [24]. That is, the server only stores the latest position data for each object. As the user locations are volatile, this setting can greatly reduce the storage and maintenance costs on the server.

The contributions of this paper are summarized as follows.

- We formulate the indoor social distance monitoring problem, and propose a framework for handling the whole monitoring process. Also, we propose an object updating mechanism that helps to reduce the server side computation.
- We analyze the indoor location uncertainty, and derive the expected distance for object pairs. Based on the analysis, we design efficient algorithms to search for the possible pairs as the results and maintain the results properly.
- We perform extensive experiments to validate the efficiency and effectiveness of our framework and algorithms.

The rest of the paper is organized as follows. Section 2 formulates the problem and presents an overall framework to address it. Section 3 elaborates on the location uncertainty model and expected distances for indoor objects. Section 4 details our computational techniques. Section 5 reports on the experimental study and Section 6 reviews the related works. Section 7 concludes the paper.

2 PROBLEM DEFINITION AND SOLUTION OVERVIEW

2.1 Indoor Positioning Data

Table 1 shows the notations used in this paper. Following the online data setting in the literature [24], we assume only the latest reporting location is stored for each moving object in an *online indoor positioning table* (OIPT). In particular, the OIPT record is in the format of (oid, l, tl), where oid identifies an object in the object set O, l is a location estimate and tl is the *latest reporting time*. For simplicity, we denote an object o_i's location estimate as l_i and the latest reporting time as tl_i throughout this paper.

Table 1: Notations

Symbol	Description
tl_i	The latest reporting time of an object o_i
$[t_c, t_f]$	The future prediction time interval
$\Delta_i(t)$	The time interval length between tl_i and t
$UR(o_i, t)$	The uncertainty region of o_i at time t
o_i^m	The m-th discrete sample of o_i
$o_i[a]$	The uncertainty sub-region of o_i
N_i^t	Number of uncertainty sub-regions of o_i at time t
$P(c_i)$	The partition that a point c_i falls in
$D(c_i)$	Set of doors that are associated to the partition $P(c_i)$

Table 2: An OIPT Example

oid	$l(c, r)$	tl	oid	$l(c, r)$	tl
o_1	$(c_1, 3)$	t_4	o_4	$(c_4, 1)$	t_4
o_2	$(c_2, 2.5)$	t_2	o_5	$(c_5, 2.5)$	t_6
o_3	$(c_3, 3.5)$	t_3	o_6	$(c_6, 2)$	t_6

Due to the limitations of indoor positioning [19], the location estimates in OIPT are usually with an error of a few meters. Following the existing studies [42, 43], we model each location estimate l_i by a circular region $\odot(c_i, r_i)$ centered at a point c_i with a radius r_i, meaning that the possible location of o_i is within $\odot(c_i, r_i)$.

In an OIPT, the latest reporting times could be different across objects since the positioning system produces the positioning records aperiodically for an object. The updating of positioning records is controlled by two system parameters, namely T_{Min} and T_{Max}, the shortest and longest time interval between two consecutive updates of an object, respectively. The former avoids overwhelming updates. The latter, on the other hand, guarantees the location estimate of an object will be updated to OIPT no more than a period of T_{Max}. Otherwise, the system would treat the object as offline and remove the corresponding record, as in the existing works [15, 32, 54].

Example 1. *A snapshot of an example OIPT at timestamp t_6 is shown in Table 2, which contains 6 objects. For example, the location estimate l_1 for o_1 is $\odot(c_1, 3)$ and its latest reporting time $tl_1 = t_4$.*

2.2 Problem Definition

Given a timestamp t, two objects o_i and o_j, let $dist(o_i, o_j, t)$ be the indoor distance from o_i to o_j at time t. The object location estimates in OIPT are already out-of-date if $tl_i, tl_j < t$. In this case, we need to analyze the possible locations of the two objects at the timestamp t in calculating $dist(o_i, o_j, t)$. The details will be given in Section 3. We define our research problem as follows.

Problem (Continuous Social Distance Monitoring). *Given a set O of indoor objects, a distance threshold ϵ, the continuous social distance monitoring (SDM) problem identifies the pairs of close contact objects along with their earliest timestamp in close contact. Formally,* *SDM constantly generates the triplet in the form of* (o_i, o_j, t) *to a result set R, satisfying* $\forall(o_i, o_j, t) \in R, o_i, o_j \in O \wedge o_i \neq o_j \wedge dist(o_i, o_j, t) \leq \epsilon$ *and* $\nexists(o_i, o_j, t') \in R, t' < t$.

We only report the first found timestamp in our setting to avoid overwhelming results from some particular close contact object pairs. Besides monitoring the distance-based condition $dist(o_i, o_j, t) \leq \epsilon$, we can easily extend our framework to consider other types of conditions such as o_i and o_j are located in the same room/floor.

2.3 Overall Framework for SDM

We propose an incremental approach for query processing, because not every object update causes changes to the query result. Specifically, when an insert/update of OIPT record arrives, we check if it will affect the query result and compute the relevant indoor distances accordingly. Consequently, the query result will be updated incrementally only based on those computed indoor distances.

At the current timestamp t_c when a new OIPT record is received, we need to identify the close contact object pairs within a short future time interval. This is necessary because such a prediction gives a reaction time for objects to avoid close contact. We define the *future prediction interval* T_{FP} as the time interval during which we monitor and predict the contacts for the OIPT records at t_c. That is, we monitor and predict the distances between objects for time interval $[t_c, t_f]$ at time t_c, where $t_f = t_c + T_{FP}$. In practice, T_{FP} is a system parameter within $[T_{Min}, T_{Max}]$. If $T_{FP} < T_{Min}$, the close contacts will only be predicted within the time interval $[t_c, t_c + T_{FP}]$, not covering $(t_c + T_{FP}, t_c + T_{Min}]$. This shortens the reaction time for objects to avoid close contacts, rendering our contact predictions less useful. In contrast, if $T_{FP} > T_{Max}$, a new OIPT record must arrive before the timestamp $t_c + T_{FP}$, which makes the prediction after receiving the new update less meaningful. Figure 1 shows an example of the three intervals, where t_c is the current timestamp. We set T_{Min} = 5s, T_{FP} = 10s and T_{Max} = 20s. This ensures, at each second, the contacts in the coming 10 seconds are being predicted, and all records are at most 20 seconds old.

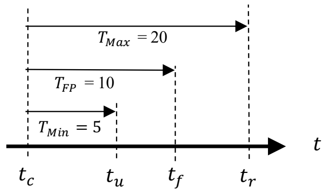


Figure 1: Example of time intervals.

Figure 2 shows the overall framework of the system. When a new query is issued in the system, we compute the initial results. The results of the query are stored in the main memory. As time passes, the system receives the location updates from objects, and the query result will be updated accordingly by the query updating/processing module. To compute concrete indoor distances, the indoor distance calculation module is called by the query updating/processing module. The main modules are described as follows.

Query Updating/Processing Module computes the initial results when a new query is issued, and updates the results when an object o_i's location is updated in OIPT. To check whether o_i is in contact with other objects, it retrieves the object locations in OIPT, checks if any of them will have contact with o_i during the prediction interval, and updates the query result accordingly. As we can compute the initial result by adapting the range join algorithm [43], we focus on how to efficiently maintain the result. The efficient update computation will be detailed in Section 4.

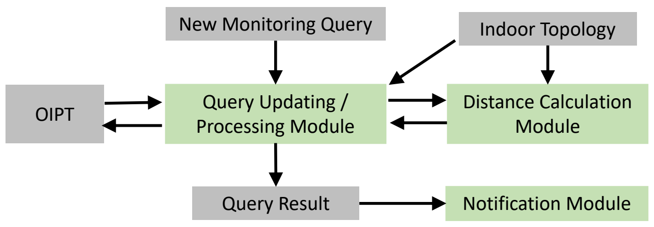


Figure 2: Continuous monitoring query processing.

Distance Calculation Module calculates the concrete distances between two objects using the uncertain and discrete location estimates in OIPT. The details will be introduced in Section 3.

Notification Module maintains the result as a list of triplets (o_i, o_j, t) in the main memory, and notifies the users if they are going to contact with others soon. If the timestamp in a triplet expires, the module removes the triplet from the list. In addition, it notifies the object to update its location when the object has been determined or suspected to be in close contact with others.

3 INDOOR DISTANCE CALCULATIONS

3.1 Indoor Space Foundations

In an indoor space, we use *partitions* to refer to rooms, staircases or hallways. They are the basic topological units in an indoor space and are connected by doors. One can go from a partition to another through their common doors. To capture the indoor topology, we use the accessibility graph [29], and the mappings of $P2D$ and $D2P$. Specifically, given a door d_i, $D2P(d_i)$ is mapped to two partitions p_i and p_j such that one can enter and leave through d_i. Inversely, given a partition p_i, $P2D(p_i)$ maps p_i to the set of doors through which one can enter or leave p_i.

Given two doors d_i and d_j, we use $|d_i, d_j|_I$ to denote the distance of the indoor shortest path from d_i to d_j. Following the existing study [43], given two indoor points p and q, we use $|q, p|_I$ to denote the distance of the indoor shortest path from q to p. Let $D(q)$ be the set of doors of the partition that q is located in. We have

$$|q, p|_I = \min_{d_q \in D(q), d_p \in D(p)} (|q, d_q|_E + |d_q, d_p|_I + |d_p, p|_E) \tag{1}$$

where $|q, d_q|_E$ and $|d_p, p|_E$ are the Euclidean distance from q to d_q and that from d_p to p, respectively. In practice, depending on the concrete layout within a partition, we can adopt other distance metrics such as obstacle distance [53] and Manhattan distance. For example, when the partition is an empty space (e.g., in a gallery hall), the Euclidean distance can be used. When the partition is an office space with rows of desks and chairs, Manhattan distance can be adopted. When the partition has furniture of irregular shapes, the obstacle distance is more appropriate. Nevertheless, our framework adapts to the aforementioned spatial distance types.

To speed up the indoor distance computation, we utilize three indexes in this paper. (1) The door-to-door distance matrix $D2D$ [29] where $D2D[i][j]$ stores $|d_i, d_j|_I$. (2) The distance index matrix

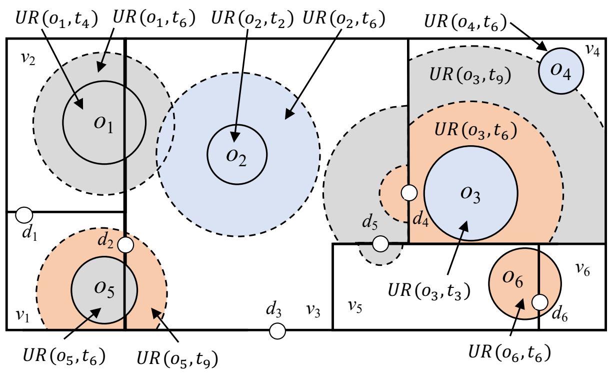


Figure 3: A running example.

$D2D_{id}$ [29], where $D2D_{id}[i][k]$ stores the ID of a door whose indoor distance from d_i is the k-th shortest among all doors. We build $D2D$ and $D2D_{id}$ following [29]. (3) The partition-to-partition dominating door matrix $P2P$, which is constructed as follows. Given two partitions p_i and p_j, we say $d_{do} \in P2D(p_i)$ is a *dominating door* if the shortest path for any point in p_i to reach p_j must pass through d_{do}. The shortest path computation utilizes the matrices $D2D$ and $D2D_{id}$. To build $P2P$, for each pair of partitions p_i and p_j, we set $P2P[i][j] = d_{do}$ if there exists a door d_{do} in $P2D(p_i)$ having the smallest distance to all doors d_j in $P2D(p_j)$. Otherwise, $P2P[i][j] = \emptyset$.

3.2 Uncertainty Regions of Moving Objects

The OIPT keeps an object o_i's location estimate $l_i = \odot(c_i, r_i)$ at the latest reporting time tl_i. For a newer timestamp $t > tl_i$, the object's location becomes further uncertain. To model o_i's possible location at time t, we adapt the *indoor uncertainty region* from a previous study [24] as follows. We expand the circular region $\odot(c_i, r_i)$ at tl_i outwardly by an indoor distance $s_{max} \cdot (t - tl_i)$, where s_{max} denotes the maximum moving speed of all indoor objects[2]. The resultant region is denoted as $UR(o_i, t)$. Unlike the study [24] that expands the uncertainty region from a single positioned point (i.e., $r_i = 0$), we expand the uncertainty region from its boundary. Since location estimates usually manifest a certain degree of positioning errors, our modeling of uncertainty regions better captures real scenarios.

Given two timestamps $t_1 > t_2 > tl_i$, it is easy to see $UR(o_i, tl_i) = \odot(c_i, r_i) \subseteq UR(o_i, t_2) \subseteq UR(o_i, t_1)$. Note that the location estimates modeled as circular regions are not affected by the underlying doors and partitions, but the expansion of indoor uncertainty regions must consider the topological restriction formed by doors and partitions.

Example 2. *Figure 3 shows 6 objects corresponding to the OIPT in Table 2. Consider the object o_1 whose location estimate is $\odot(c_1, 3)$ at the latest reporting time $tl_1 = t_4$. Its uncertainty region $UR(o_1, t_4)$ is the circle with solid circumference that overlaps partitions v_2 and v_3. For a newer time $t_6 > t_4$, its uncertainty region $UR(o_1, t_6)$ expands to the concentric circle enclosed by the dashed circumference.*

[2]E.g., s_{max} can be the maximum walking speed when the objects are people. While s_{max} is the same for all objects in this paper, our framework can support individualized s_{max}s for different objects.

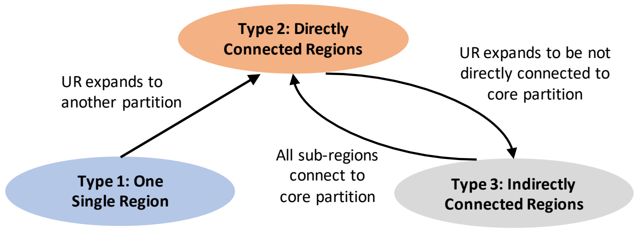

Type 2: Directly Connected Regions
UR expands to another partition
UR expands to be not directly connected to core partition
All sub-regions connect to core partition
Type 1: One Single Region
Type 3: Indirectly Connected Regions

Figure 4: Indoor uncertain object type diagram.

Given an object o_i, we use *core partition* to denote the partition that c_i is located in, and *uncertainty sub-regions* to refer to the different portions of an uncertainty region inside different partitions. Accordingly, we differentiate three types of objects:

Type 1 (One Single Region): The object and its uncertainty region lie in the core partition only. In Figure 3, o_2 and o_4 are of type 1 at time t_6.

Type 2 (Directly Connected Regions): The object's uncertainty region spans multiple partitions, and all sub-regions are connected to the core partition by doors. In Figure 3, o_6 is of type 2 at time t_6, since its uncertainty sub-regions overlap with partitions v_5 and v_6, and they are connected by door d_6.

Type 3 (Indirectly Connected Regions): The object's uncertainty region spans multiple partitions, while there exist at least one sub-region that is not directly connected to the core partition by any door. In Figure 3, o_1 and o_5 are of type 3 at time t_6. For either of them, its uncertainty sub-regions are not directly connected.

Object types enable us to design different pruning techniques for computing and bounding the distances between different types of objects, as to be discussed in Section 4.1.

We then discuss how object types may vary as the time elapses and uncertainty regions expand. Figure 4 depicts the transitions, for which the conditions are summarized as follows.

- When a type 1 object expands its uncertainty region to another partition, it becomes a type 2 object (e.g., o_3 at time t_6).
- When a type 2 object expands its uncertainty region to a partition that is indirectly connected to the core partition, it becomes a type 3 object (e.g., o_3 has a part of its uncertainty region in v_5 at time t_9, which is indirectly connected to the core partition v_4).
- When a type 3 object expands its uncertainty region such that all sub-regions connected to the core partition directly, it becomes a type 2 object (e.g., the two sub-regions of o_5 connect through door d_2 at time t_9).

Example 3. *Figure 5 shows the temporal view of three objects. Consider o_3. At time t_3, it is of type 1, as its uncertainty region is only located inside v_4. At time t_6, it is of type 2, as its uncertainty region overlaps with both v_3 and v_4. At time t_9, it becomes of type 3, as its uncertainty sub-region inside v_5 is indirectly connected to v_4.*

Algorithm 1 shows the procedure of determining an object o_i's types in its life time. It returns a list of tuples which each specifies a time interval for its type. Specifically, it maintains the variables *curType* and *nextType* storing the types of o_i in the current and next intervals, respectively. It first finds o_i's initial type based on the current location estimate (line 1), and performs an iterative process to determine the duration for each interval (lines 2 to 14). Consider an iteration. It determines the duration of *curType* as

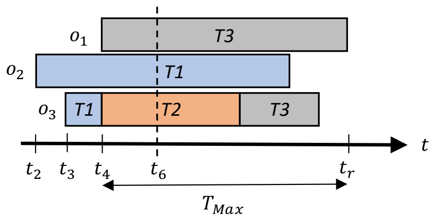


Figure 5: Temporal view of objects o_1, o_2 and o_3 ($t_c = t_4$).

follows. If *curType* = *TYPE1*, it finds the minimum time needed for o_i to expand its uncertainty region to another partition, and sets *nextType* to *TYPE2* (lines 3 to 5). If *curType* = *TYPE2*, it finds the minimum time for o_i to expand to a partition that is indirectly connected to its core partition, and sets *nextType* to *TYPE3* (lines 6 to 8). If *curType* = *TYPE3*, it finds the minimum duration needed for all sub-regions to connect to the core partition directly, and sets *nextType* to *TYPE2* (lines 9 to 11). If such a case does not exist, the duration is set to T_{Max}. The interval is then inserted into the list *Types*, and the iteration is repeated until all types' intervals have been determined. Finally, *Types* is returned as the result.

Algorithm 1 `findObjTypes`(o_i)

```
1: curType ← get initial type of $o_i$; $t_s \leftarrow tl_i$; $t_e \leftarrow tl_i$
2: while $t_s < tl_i + T_{Max}$ do
3:     if curType = TYPE1 then
4:         $minD \leftarrow \min_{d \in D(P(c_i))} |c_i, d|_E$; $\Delta t \leftarrow (minD - r_i)/s_{max}$
5:         nextType ← TYPE2
6:     else if curType = TYPE2 then
7:         $\Delta t$ ← duration to expand to an indirectly connected partition
8:         nextType ← TYPE3
9:     else
10:        $\Delta t$ ← duration for all sub-regions to connect directly
11:        nextType ← TYPE2
12:    if $\Delta t > T_{Max}$ then $t_e \leftarrow tl_i + T_{Max}$ else $t_e \leftarrow t_s + \Delta t$
13:    Types.insert(curType, $[t_s, t_e)$)
14:    curType ← nextType; $t_s \leftarrow t_e$
15: return Types
```

3.3 Expected Indoor Distance

Consider an uncertain object o_i at time t. The exact location of o_i can be modeled as a random variable l associated with a *probability density function* (pdf). The pdf can be represented by either a closed form equation [8, 9], or a set of discrete samples [25, 43]. In this paper, we adopt the discrete sample representation, as it has the advantage of modeling arbitrary distributions, such as the distance decay functions (DDFs) [24]. That is, given a moving object o_i at time t, we have a set of pairs $\{(o_i^m, o_i^m.\rho_t)\}$, where o_i^m is the m-th sample located inside $UR(o_i, t)$ and $o_i^m.\rho_t$ is its existential probability at time t, satisfying $\sum_{o_i^m \in o_i} o_i^m.\rho_t = 1$.

Sample-based representation can be generated by many indoor object movement prediction models such as Hidden Markov Models (HMM) [31, 34, 37], Mixed Markov Models (MMM) [3], particle filters [11, 12], and Bayesian Networks [33]. Thus, those models can be used to determine the existential probabilities of the samples.

As the moving objects can be in anywhere in their corresponding uncertainty region, the inter-object distance is probabilistic in nature. Thus, we utilize the expected indoor distance to measure the distance. Based on the discrete sample representation, we define the expected indoor distance for moving objects as follows.

Definition 1 (Expected indoor distance [43]). *Given a timestamp t and two moving objects o_i and o_j, the expected indoor distance between o_i and o_j at t is:*

$$dist(o_i, o_j, t) = E_{o_i^m \in UR(o_i,t), o_j^n \in UR(o_j,t)}(|o_i^m, o_j^n|_I)$$
$$= \sum_{o_i^m \in UR(o_i,t)} \sum_{o_j^n \in UR(o_j,t)} |o_i^m, o_j^n|_I \cdot o_i^m.\rho_t \cdot o_j^n.\rho_t \quad (2)$$

Above, the number of samples in o_i and o_j are proportional to the area of their uncertainty regions. Note that the expected indoor distance defined in [43] only considers the samples from static uncertainty regions. In contrast, our definition on the expected distance is time-parameterized such that it can estimate the distances for two moving objects with expanding uncertainty regions.

Based on the types of the object pair, we use different formulas to calculate the $dist(o_i, o_j, t)$.

(I) Type 1 Object Pairs. When both o_i and o_j are type 1 objects, we have the following three cases.

- **Case 1** (both objects are located at the same partition). In this case, all samples are in the same partition. We simply use the Euclidean distance to compute $dist(o_i, o_j, t)$ as

$$\sum_{o_i^m \in UR(o_i,t)} \sum_{o_j^n \in UR(o_j,t)} |o_i^m, o_j^n|_E \cdot o_i^m.\rho_t \cdot o_j^n.\rho_t \quad (3)$$

- **Case 2** (all samples of o_i have the same door sequence to all samples of o_j). In this case, we can reuse the shortest distance information for all pairs and compute $dist(o_i, o_j, t)$ as

$$\sum_{o_i^m \in UR(o_i,t)} |o_i^m, d_i|_E \cdot o_i^m.\rho_t + |d_i, d_j|_I + \sum_{o_j^n \in UR(o_j,t)} |d_j, o_j^n|_E \cdot o_j^n.\rho_t \quad (4)$$

where d_i and d_j are the doors of the partition containing o_i and o_j, respectively.

- **Case 3** (otherwise). In this case, we need to consider all pairs of samples and compute $dist(o_i, o_j, t)$ as

$$\sum_{o_i^m \in UR(o_i,t)} \sum_{o_j^n \in UR(o_j,t)} |o_i^m, o_j^n|_I \cdot o_i^m.\rho_t \cdot o_j^n.\rho_t \quad (5)$$

(II) Object Pairs Involving Type 2 or Type 3 Object. Let N_i^t be the number of uncertainty sub-regions that o_i has at time t. We have $o_i = \cup_{1 \le a \le N_i^t} o_i[a]$, where each $o_i[a]$ corresponds to the sub-region in a different partition and it consists of a set of samples. In this case, we calculate the distance separately for each pair of sub-regions, using the above Equations 3, 4 or 5. Specifically, with a slight abuse of notation, we compute $dist(o_i, o_j, t)$ as

$$\sum_{a=1}^{N_i^t} \sum_{b=1}^{N_j^t} \left(dist(o_i[a], o_j[b], t)\right) \quad (6)$$

Based on the above discussion, we present `distCalc` function (formalized in Algorithm 2) that calculates the expected indoor distance between o_i and o_j at a time t. Specifically, it first checks whether both o_i and o_j are type 1 objects. If so, it computes the distances between o_i and o_j according to the case they belong to

Algorithm 2 distCalc(o_i, o_j, t)

```
1:  $dist \leftarrow 0$
2:  if $N_i^t = 1$ and $N_j^t = 1$ then                              ▹ Type 1 Object Pair
3:      $p_i \leftarrow P(o_i); p_j \leftarrow P(o_j)$
4:      if $p_i = p_j$ then                                                  ▹ Case 1
5:          for each sample pair $(o_i^m, o_j^n) \in UR(o_i, t) \times UR(o_j, t)$ do
6:              $dist \leftarrow dist + |o_i^m, o_j^n|_E \cdot o_i^m.\rho_t \cdot o_j^n.\rho_t$
7:      else if $P2P[i][j] \neq \emptyset$ and $P2P[j][i] \neq \emptyset$ then       ▹ Case 2
8:          $d_i \leftarrow P2P[i][j]; d_j \leftarrow P2P[j][i]$
9:          $dist \leftarrow dist + |d_i, d_j|_I$
10:         for each sample $o_i^m \in UR(o_i, t)$ do
11:             $dist \leftarrow dist + |o_i^m, d_i|_E \cdot o_i^m.\rho_t$
12:         for each sample $o_j^n \in UR(o_j, t)$ do
13:             $dist \leftarrow dist + |d_j, o_j^n|_E \cdot o_j^n.\rho_t$
14:     else                                                                 ▹ Case 3
15:         for each sample pair $(o_i^m, o_j^n) \in UR(o_i, t) \times UR(o_j, t)$ do
16:             $dist \leftarrow dist + |o_i^m, o_j^n|_I \cdot o_i^m.\rho_t \cdot o_j^n.\rho_t$
17: else                          ▹ Object Pair Involving Type 2 or Type 3 Object
18:     for each sub-region pair $(o_i[a], o_j[b])$ do
19:         $dist \leftarrow dist +$ distCalc($o_i[a], o_j[b], t$)
20: return $dist$
```

(lines 2–16). In Case 2, the $P2P$ index and pre-computed door-to-door distances (see Section 3.1) are used to accelerate the computation (lines 8–9). Otherwise, it computes the distance by summing up the distances for each sub-region pair (obtained by calling distCalc(·)). Finally, it returns $dist$ as the result.

As object types change with time, it is possible that the formula used for calculating $dist(o_i, o_j, t)$ changes with time. In any case, we need to find the shortest distance for every pair of samples from o_i and o_j, which requires $O(m \cdot n \cdot T_{FP})$ shortest distance calculations for each object pair. In Section 4, we will discuss effective pruning techniques and algorithms to avoid expensive computations.

4 COMPUTATION APPROACH

In this section, we present the computation approach employed in the query updating/processing module for maintaining the result continuously as objects are updated. In particular, we propose several effective pruning techniques in Section 4.1, and based on these techniques, we develop an efficient query updating algorithm and batch processing algorithm in Sections 4.2 and 4.3, respectively.

4.1 Pruning

In fact, not all object pairs at all timestamps in the prediction interval need to be considered. In the following, we consider floor-based, topology-based, and probability-based prunings to reduce the number of candidate object pairs for which we need to compute $dist(o_i, o_j, t)$.

4.1.1 *Floor-based Pruning.*

We lower bound the expected distance between the object pairs that are on different floors. In fact, staircases are crucial in determining the bound, because an object must go through one of the staircases to reach the other floors. If the object is far away from the staircases, it cannot visit and contact the objects on other floors under the maximum speed constraint. Let c_i and c_j be the center point of l_i and l_j, respectively, the skeleton distance [43] is computed as

$$|c_i, c_j|_K = \begin{cases} |c_i, c_j|_E, & \text{if } c_i.f = c_j.f; \\ \min_{\substack{sc_i \in SC(c_i.f),\\ sc_j \in SC(c_j.f)}} \Big(|c_i, sc_i|_E + \\ \quad |sc_i, sc_j|_I + |sc_j, c_j|_E\Big), & \text{otherwise.} \end{cases}$$

where $SC(f)$ is the set of staircase doors in the f-th floor. It was proved in the study [43] that $|c_i, c_j|_K \leq |c_i, c_j|_I$. We utilize this definition and give the following lemma for pruning.

Lemma 1 (Time-parameterized Skeleton Distance Bound). *Given two objects o_i, o_j and a time t, let $\Delta_i(t) = t - tl_i$ and $\Delta_{ij}(t) = \Delta_i(t) + \Delta_j(t)$, we have the following skeleton distance bound function.*

$$dist(o_i, o_j, t)_{LB} = |c_i, c_j|_K - r_i - r_j - s_{max} \cdot \Delta_{ij}(t) \tag{7}$$

Proof. As proved in the study [43] that $d(o_i, o_j, t) \geq |c_i, c_j|_I - r_i - r_j$ for $t = tl_i = tl_j$. Thus, $d(o_i, o_j, t) \geq |c_i, c_j|_K - r_i - r_j$ as $|c_i, c_j|_I \geq |c_i, c_j|_K$. As the maximum moving speed is s_{max}, we have that in the worst case o_i can move $s_{max} \cdot \Delta_i(t)$ since tl_i. The same applies to o_j. Combining the above, we have $dist(o_i, o_j, t) \geq |c_i, c_j|_K - r_i - r_j - s_{max}(\Delta_i(t) + \Delta_j(t)) = dist(o_i, o_j, t)_{LB}$. □

Consider an object o_i located on the floor $o_i.f$. Based on the above lemma, we propose to filter the objects in OIPT that cannot contribute to a contact pair with o_i as follows. Let $|c_i, sd_k|_I$ be the minimum indoor distance from c_i to one of the staircases' doors in $o_i.f$, and $lenSC$ be the length of a stairway (i.e., the distance between two adjacent floors). If $|c_i, sd_k|_I + r_i + 2 \cdot s_{max} \cdot \Delta_i(t) + \epsilon < lenSC \cdot |o_i.f - f'|$, where $|o_i.f - f'|$ is the floor number difference between $o_i.f$ and f', we do not need to process the objects on the f'-th floor at time t, as the objects on f'-th floor are too far away to reach o_i. In practice, this simple pruning is effective since it can restrict the search space of processing o_i to o_i's nearby floors only.

4.1.2 *Topology-based Pruning.*

Given an object o_i and some other objects $o_j \in O$, the topology-based pruning removes those candidate object pairs that have $dist(o_i, o_j, t) > \epsilon$ for some timestamps t, and includes the pairs that must be in contact within the prediction interval. In other words, we want to identify those tuples (o_i, o_j, t) such that o_i and o_j always have their expected distance to each other greater or smaller than ϵ at time t.

Specifically, we consider the topological distance bounds between each object pair. For different types of objects, different pruning bounds are applied. We first consider the simple case that both o_i and o_j are type 1 objects (i.e., both objects' uncertainty regions only overlap with one partition). Inspired by the study [43], we develop the following lemma.

Lemma 2 (Time-parameterized Topological Distance Bounds for Type 1 Object Pairs). *Given two type 1 objects o_i, o_j and a timestamp t, we have the following distance bounds for $dist(o_i, o_j, t)$.*

$$dist(o_i, o_j, t)_{LB} = |c_i, c_j|_I - r_i - r_j - s_{max} \cdot \Delta_{ij}(t)$$
$$dist(o_i, o_j, t)_{UB} = |c_i, c_j|_I + r_i + r_j + s_{max} \cdot \Delta_{ij}(t)$$
$$s.t.\ dist(o_i, o_j, t)_{LB} \leq dist(o_i, o_j, t) \leq dist(o_i, o_j, t)_{UB}$$

Proof. As proved in the study [43] that $d(o_i, o_j, t) \geq |c_i, c_j|_I - r_i - r_j$ for $t = tl_i = tl_j$. We proceed to prove $dist(o_i, o_j, t) \geq$

$|c_i, c_j|_I - r_i - r_j - s_{max} \cdot \Delta_{ij}(t)$ for all ts. Given the maximum speed s_{max}, we know that in the worst case o_i can move $s_{max} \cdot \Delta_i(t)$ since tl_i. The same applies to o_j. Thus, we have

$$dist(o_i, o_j, t) \geq |c_i, c_j|_I - r_i - r_j - s_{max} \cdot \Delta_{ij}(t) = dist(o_i, o_j, t)_{LB}$$

The case for $dist(o_i, o_j, t)_{UB}$ is similar and thus is omitted due to the page limit. □

As $dist(o_i, o_j, t)_{LB}$ (resp. $dist(o_i, o_j, t)_{UB}$) is monotonic decreasing (resp. increasing) as time passes, we find the timestamp t_{LB} (resp. t_{UB}) that $dist(o_i, o_j, t)_{LB} = \epsilon$ (resp. $dist(o_i, o_j, t)_{UB} = \epsilon$). Based on the relationship between t_{LB}, t_{UB}, t_c and t_f, we discuss the following cases.

- **Case 1** ($t_{LB} > t_f$): We can safely prune this pair as we have $dist(o_i, o_j, t) > \epsilon$ for any time $t \in [t_c, t_f]$.
- **Case 2** ($t_{LB} < t_c$): It is possible that $dist(o_i, o_j, t) \leq \epsilon$ for $t \in [t_c, t_f]$. There are two sub-cases.
  - **(a)** ($t_{UB} \geq t_c$): We simply insert the tuple (o_i, o_j, t_c) into the result, since we know that $dist(o_i, o_j, t_c) < \epsilon$, and we terminate the search on this pair.
  - **(b)** ($t_{UB} < t_c$): We add the pair as a candidate pair and check the distances in the interval $[t_c, t_f]$.
- **Case 3** ($t_{LB} \in [t_c, t_f]$): We separate the time interval into two sub-intervals. The interval $[t_c, t_{LB}]$ can be safely pruned, similar to Case 1. The interval $[t_{LB}, t_f]$ needs to be handled in a way similar to Case 2. Figure 6 shows an example in this case.

We proceed to consider the bounds for a pair of type 2 objects. In fact, we can easily generalize the bounds in Lemma 2 by modifying the definition of r_i to be the maximum indoor distance among all distances from c_i to the boundary of the uncertainty regions.

For a pair of type 3 objects, we treat each uncertainty sub-region of an object as a type 1 object and utilize Lemma 2, with the following adaption. Consider a sub-region $o_i[a]$. Let $P(o_i[a])$ denotes the partition in which $o_i[a]$ is located in. We create a fictitious center c'_i with the same position as c_i (but still located inside $P(o_i[a])$), and set the radius r_i to be the Euclidean distance from c_i to the boundary of $o_i[a]$. An example is shown in Figure 7, where the sub-region $o_i[2]$ is located at v_2 with the center c'_i. While c'_i is actually outside $o_i[2]$, we need to connect it to the doors of $P(o_i[2])$ first when computing its indoor distances to the points in other partitions. Then, we have the following lemma.

Lemma 3 (Time-parameterized Topological Distance Bounds for Type 3 Object Pairs). *Given two type 3 objects o_i, o_j and a timestamp t, we have the following distance bounds for $dist(o_i, o_j, t)$.*

$$dist(o_i, o_j, t)_{LB} = \min_{1 \leq a \leq N_i^t, 1 \leq b \leq N_j^t} dist(o_i[a], o_j[b], t)_{LB}$$

$$dist(o_i, o_j, t)_{UB} = \max_{1 \leq a \leq N_i^t, 1 \leq b \leq N_j^t} dist(o_i[a], o_j[b], t)_{UB}$$

$$s.t.\ dist(o_i, o_j, t)_{LB} \leq dist(o_i, o_j, t) \leq dist(o_i, o_j, t)_{UB}$$

Proof. We first prove the case for $dist(o_i, o_j, t)_{LB}$. Consider a sub-region $o_i[a]$. If $P(c_i) = P(o_i[a])$, it is similar to the proof in Lemma 2. Otherwise, we prove as follows. Let $D(o_i[a])$ be the set of doors associated with $P(o_i[a])$, d_i be a door in $D(o_i[a])$, and x be a point in $o_i[a]$. By the triangle inequality, we have $|c'_i, d_i|_E - |c'_i, x|_E \leq |x, d_i|_E$. Also, we know that $|c'_i, x|_E \leq r_i$. We then have $|c'_i, d_i|_E - r_i \leq |x, d_i|_E$. Thus, $dist(o_i[a], o_j[b], t)_{LB} \leq$

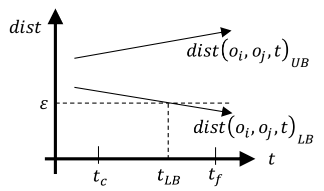


Figure 6: Time-parameterized topological bounds.

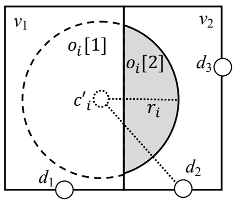


Figure 7: The sub-region $o_i[2]$ and its fictitious center c'_i.

$dist(x, o_j[b], t)$ for all points $x \in o_i[a]$. The same applies for $o_j[b]$. The case for $dist(o_i, o_j, t)_{UB}$ is similar, but utilizing another triangle inequality $|c'_i, d_i|_E + |c'_i, x|_E \geq |x, d_i|_E$. □

So far, Lemmas 2 and 3 handle a pair of objects that are of static and identical types. To make the lemmas generic to all object pairs, we discuss the following two extensions, namely object type change and different object types in a pair.

Object Type Change. As we discussed in Section 3.2, the type of an object could change during its lifetime. In this case, we simply perform separated checking for each sub-interval segmented by the timestamp(s) when an object changes its type. Suppose that an object changes its type twice, from type 1 to type 2 at a time $t_{e1} \in [t_c, t_f]$, and from type 2 to type 3 at a time $t_{e2} \in [t_c, t_f]$. Then, we perform the pruning for sub-intervals $[t_c, t_{e1})$, $[t_{e1}, t_{e2})$ and $[t_{e2}, t_f]$, separately.

Different Object Types in a Pair. Our pruning lemmas can be easily extended to bound the distances for the object pair with different types. For example, when o_i is a type 3 object and o_j is a type 1 object at time t, we have the following distance lower bound.

$$dist(o_i, o_j, t)_{LB} = \min_{1 \leq a \leq N_i^t} dist(o_i[a], o_j, t)_{LB}$$

The distance of the object pairs with other different types can be bounded in a similar way. It is easy to see that the correctness of the bounds is not affected. We omit the complete list of bounds with all possible type combinations here due to the space limit.

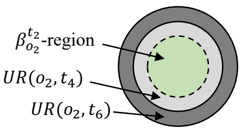
4.1.3 *Probability-based Pruning.*

We consider the probability distribution among the samples of objects. First, we introduce the concept of β-region.

Definition 2 (β-region [26, 27]). *Given an object o_i, the β-region of o_i is a closed region such that o_i locates inside this region with a probability at least β.*

Recall that we defined $o_i^m.\rho_t$ as the existential probability of the sample o_i^m at a timestamp t in Section 3.3. The β-region is constructed as follows. Given a pre-defined region $\Phi \subseteq UR(o_i, t)$ for a timestamp t, we set β to be the sum of probabilities of discrete samples inside Φ, i.e., $\beta = \sum_{o_i^m \in \Phi} o_i^m.\rho_t$. We use a circle to be the shape of Φ, with center c_i, and radius being the maximum distance from c_i of the samples. We extend the above definition to the time-parameterized β-region.

Definition 3 (Time-parameterized $\beta(t)$-region). *Given an object o_i and a timestamp $t \in [t_c, t_f]$, the time-parameterized $\beta(t)$-region of o_i is a closed region such that o_i locates inside this region with a probability at least $\beta(t)$ at timestamp t.*



Region	t_2	t_4	t_6	$\cdots$
β^{t_2}	1	0.8	0.6	$\cdots$
β^{t_4}	-	1	0.7	$\cdots$
β^{t_6}	-	-	1	$\cdots$

Figure 8: An example of $\beta^{t_c}_{o_2}$-region.

We construct a $\beta(t)$-region to be the region that is exactly the same region as $UR(o_i, t_p)$ for each timestamp $t_p \in [t_c, t_f]$, denoted by $\beta_i^{t_p}(t)$. When the uncertainty region of o_i expands with t, it is easy to see that the probability $\beta_i^{t_p}(t)$ decreases accordingly.

Example 4. *An example of the $\beta(t)$-region of object o_2 is shown in Figure 8. Given the $\beta_2^{t_2}(t)$-region (which is equal to $UR(o_2, t_2)$), we have $\beta_2^{t_2}(t_4) = 0.8$ and $\beta_2^{t_2}(t_6) = 0.6$.*

We utilize the above definition and derive the following lemma.

Lemma 4 (Time-parameterized β-region bounds). *Given two objects o_i, o_j, a timestamp t and a timestamp $t_p < t$, we have*

$$\begin{aligned} LB_\beta(o_i, o_j, t) &= \beta_i^{t_p}(t)\beta_j^{t_p}(t) \cdot dist(o_i, o_j, t_p) \\ &\quad + (1 - \beta_i^{t_p}(t)\beta_j^{t_p}(t))\ dist(o_i, o_j, t)_{LB} \\ UB_\beta(o_i, o_j, t) &= \beta_i^{t_p}(t)\beta_j^{t_p}(t) \cdot dist(o_i, o_j, t_p) \\ &\quad + (1 - \beta_i^{t_p}(t)\beta_j^{t_p}(t))\ dist(o_i, o_j, t)_{UB} \\ s.t.\ LB_\beta(o_i, o_j, t) &\le dist(o_i, o_j, t) \le UB_\beta(o_i, o_j, t) \end{aligned}$$

where $dist(o_i, o_j, t)_{LB}$ and $dist(o_i, o_j, t)_{UB}$ are any lower and upper bounds of $dist(o_i, o_j, t)$, respectively.

Proof. We prove the case for LB_β below. From Equation 2,

$$\begin{aligned} dist(o_i, o_j, t) &= E_{o_i^m \in UR(o_i,t), o_j^n \in UR(o_j,t)}(|o_i^m, o_j^n|_I) \\ &= E_{o_i^m \in UR(o_i,t_p), o_j^n \in UR(o_j,t_p)}(|o_i^m, o_j^n|_I) \\ &\quad + E_{\substack{o_i^m \in UR(o_i,t)\setminus UR(o_i,t_p),\\ o_j^n \in UR(o_j,t)\setminus UR(o_j,t_p)}}(|o_i^m, o_j^n|_I) \\ &\ge \beta_i^{t_p}(t)\beta_j^{t_p}(t) \cdot dist(o_i, o_j, t_p) \\ &\quad + (1 - \beta_i^{t_p}(t)\beta_j^{t_p}(t))\ dist(o_i, o_j, t)_{LB} = LB_\beta \end{aligned}$$

The case for UB_β is similar and is omitted due to the page limit. □

To ease the computation, we simply derive $dist(o_i, o_j, t)_{LB}$ and $dist(o_i, o_j, t)_{UB}$ in Lemma 4 based on the maximum moving speed constraint, i.e.,

$$\begin{aligned} dist(o_i, o_j, t)_{LB} &= dist(o_i, o_j, t_p) - 2 \cdot s_{max} \cdot (t - t_p) \\ dist(o_i, o_j, t)_{UB} &= dist(o_i, o_j, t_p) + 2 \cdot s_{max} \cdot (t - t_p) \end{aligned} \tag{8}$$

Consequently, we have

$$LB_\beta(o_i, o_j, t) = dist(o_i, o_j, t_p) - (1 - \beta_i^{t_p}(t)\beta_j^{t_p}(t))\ 2 \cdot s_{max}(t - t_p)$$

$$UB_\beta(o_i, o_j, t) = dist(o_i, o_j, t_p) + (1 - \beta_i^{t_p}(t)\beta_j^{t_p}(t))\ 2 \cdot s_{max}(t - t_p)$$

Based on the above lemma, we can use the distance computed at the current timestamp to bound the distances for the future timestamps. Specifically, after obtaining $dist(o_i, o_j, t_p)$ at timestamp t_p, we bound the distances $dist(o_i, o_j, t)$ for all future timestamps $t \in (t_p, t_f]$. For each such timestamp t whose lower bound distance is larger than ϵ, its distance calculation can be skipped safely. If the distance is always smaller than ϵ during $(t_p, t_f]$, the pair can be safely pruned. Similarly, if the upper bound distance at t is at most ϵ, we know that the triplet (o_i, o_j, t) must be in the result.

4.2 Query Update for One Object

We use the bounds and lemmas proposed above to efficiently generate the updated result of SDM. When an object o_i updates/inserts, we first determine the object's type. Second, depending on the type of object, we utilize the proposed pruning lemmas to remove the pairs that are not possible to be in the result. Third, for the remaining object pairs, we compute their indoor distances and insert the contact pairs to the query results.

The `queryUpdate` algorithm is presented in Algorithm 3. Given the updated location of o_i, it returns the set of contact pairs involving o_i. First, it filters the objects in OIPT that cannot contribute to the result (line 2). This is done by examining the objects' floors and their skeleton distances (see Section 4.1.1). The qualified objects are inserted into O'. Then, it checks the objects $o_j \in O'$ to see whether a contact pair (o_i, o_j) can be found (lines 3-6). For each o_j, it invokes the `checkPair` algorithm. If a triplet is returned, it is added to R. Finally, the algorithm returns R.

The `checkPair` algorithm, which checks if a candidate object pair (o_i, o_j) is in contact in $[t_c, t_f]$, is presented in Algorithm 4. First, it invokes the `findULBTime`$(\cdot)$ to find the lower and upper bound times (t_{LB} and t_{UB}, respectively) that the pair is going to contact (line 1). The procedure follows the discussion in Section 4.1.2. If $t_{LB} > t_f$, the pair cannot be in contact, and thus it returns $\emptyset$ immediately. If $t_{UB} \ge t_c$, the pair must be in contact at t_c, and thus it returns the triplet (o_i, o_j, t_c). Otherwise, it checks for each timestamp $t \in [t_s, t_f]$ (lines 5-10). For each t, it obtains LB_β and UB_β by invoking `betaBounds`$(\cdot)$. This procedure computes the lower and upper distance bounds of the candidate pair based on the discussion in Section 4.1.3. If the candidate pair is not pruned, the $dist(o_i, o_j, t)$ is computed and the triplet (o_i, o_j, t) is returned if $dist(o_i, o_j, t) \le \epsilon$ (line 9). The algorithm returns $\emptyset$ if no contact is found.

Algorithm 3 `queryUpdate`$(o_i, t_c, \text{OIPT}, \epsilon)$

```
1: $t_f \leftarrow t_c + T_{FP}$
2: $O' \leftarrow$ filterObjs$(o_i, \text{OIPT}, t_c, t_f, \epsilon)$          ▷ Floor-based Pruning
3: for each object $o_j \in O'$ do
4:     $triplet \leftarrow$ checkPair$(o_i, o_j, t_c, t_f, \epsilon)$
5:     if $triplet \ne \emptyset$ then
6:         $R \leftarrow R \cup \{triplet\}$
7: return $R$
```

Algorithm 4 `checkPair`$(o_i, o_j, t_c, t_f, \epsilon)$

```
1: $(t_{LB}, t_{UB}) \leftarrow$ findULBTime$(o_i, o_j, \epsilon)$          ▷ Topology-based Pruning
2: if $t_{LB} > t_f$ then return $\emptyset$
3: if $t_{UB} \ge t_c$ then return $(o_i, o_j, t_c)$
4: $t_s \leftarrow \max\{t_c, t_{LB}\}$
5: for $t \in [t_s, t_f]$ do
6:     $(LB_\beta, UB_\beta) \leftarrow$ betaBounds$(o_i, o_j, t)$▷ Probability-based Pruning
7:     if $LB_\beta > \epsilon$ then return $\emptyset$
8:     if $UB_\beta \le \epsilon$ then return $(o_i, o_j, t)$
9:     if $dist(o_i, o_j, t) \le \epsilon$ then return $(o_i, o_j, t)$
10: return $\emptyset$
```

4.3 Query Update based on Batch Processing

The above discussion handles the case that one object is updated at each timestamp. We extend our techniques to handle multiple object updating and inserting efficiently in a timestamp.

Consider a timestamp that there are multiple updating objects $objBatch = \{o_1, o_2, \ldots, o_l\}$, we process the objects in *objBatch* in batch by the `batchUpdate` algorithm. It has four major steps.

- **Step 1** (New Object Processing) finds the contact object pairs among the objects in *objBatch* and stores them in *R*.
- **Step 2** (Object Grouping) assigns the objects in *objBatch* into different groups based on the result set *R*.
- **Step 3** (Group Processing) processes each group with the following two sub-steps to find the contact pairs: (i) finds the candidate partitions and (ii) processes the active objects in each candidate partition.
- **Step 4** (Object Inserting) inserts all objects in *objBatch* into OIPT for future processing timestamps.

The batch processing algorithm is shown in Algorithm 5. Next, we present the details of Steps 1-3 (note that Step 4 is straightforward). In Step 1, it processes the object pairs among the new objects in *objBatch* using the `queryUpdate` algorithm (lines 1-2 in Algorithm 5). In Step 2, it puts the objects in *objBatch* into different groups heuristically as follows. If two objects o_x, o_y are located at the same partition, or if they have $dist(o_x, o_y, t) \leq \epsilon$ for any $t \in [t_c, t_f]$, we put them into a group. A group G can be viewed as a "big" object with center c_G equals to the mean of the object centers in that group, and the radius $r_G = \max_{o_i \in G}(|c_G, c_i|_E + r_i)$. We process the objects sequentially and obtain groups of close objects.

Step 3(i) processes each group to find candidate partitions. For each group G_k, it finds the candidate partitions by considering the doors that connect to the partitions. Specifically, it finds all doors d_j such that $|d_i, d_j|_I \leq 2 \cdot s_{max} \cdot T_{Max} + \epsilon + r_{max} + r_G$, where $d_i \in D(G_k)$ is a door associated with the partitions that overlap with G_k, and r_{max} is the maximum radius among all objects in the OIPT. Then, the partitions that are associated with such d_js are the candidate partitions. It can be proven that objects not in these candidate partitions cannot form contact pairs with the objects in the group. The objects in the candidate partitions form the set O' of candidate objects (lines 6-7). To enable this pruning, we maintain, for each partition p, the pointers to the objects that each has its location estimate overlapping with p. This can be done efficiently when the object is inserted or removed.

In Step 3(ii), it processes each group with the candidate objects o_j in O', and finds the resulting pairs. Since the objects in the group might have different types, we use the loosest bounds (Lemma 3) to determine $dist(G, o_j, t)_{LB}$ and $dist(G, o_j, t)_{UB}$ (lines 8-14 in Algorithm 5). If the group is not pruned by the bounds, the objects in the group are then processed one by one (lines 16-19). We evaluate the effect of the batch processing strategy in our experiments.

4.4 Complexity Analysis

We first analyze the time complexity of Algorithm 4. Let m and n be the maximum number of discrete samples of o_i and o_j, respectively. Since we need to calculate $O(m \cdot n)$ indoor distances in the worst case, its time complexity is $O(|T_{FP}| \cdot m \cdot n)$. Note that the actual number of such computations is much smaller given our topology-based

Algorithm 5 `batchUpdate`($objBatch$, t_c, OIPT, ϵ)

```
1:  for each object $o_i \in objBatch$ do                                   ▷ Step 1
2:      $R \leftarrow R \cup$ queryUpdate($o_i, t_c, objBatch \setminus \{o_i\}, \epsilon$)
3:  $\mathcal{G} \leftarrow$ grouping($objBatch, R$)                          ▷ Step 2
4:  $d2d_{UB} \leftarrow 2 \cdot s_{max} \cdot T_{Max} + \epsilon + r_{max}$
5:  for each group $G_k \in \mathcal{G}$ do
6:      $D' \leftarrow \{d_j \mid d_i \in D(G_k) \wedge |d_i, d_j|_I \leq d2d_{UB} + r_G\}$   ▷ Step 3(i)
7:      $O' \leftarrow \{o \mid o.l \cap p \neq \emptyset \wedge d_j \in D' \wedge p \in D2P(d_j)\}$
8:      for each object $o_j \in O'$ do                                     ▷ Step 3(ii)
9:          $(t_{LB}, t_{UB}) \leftarrow$ findULBTime($G_k, o_j, \epsilon$)
10:         if $t_{LB} > t_f$ then return $\emptyset$
11:         if $t_{UB} \geq t_c$ then
12:             for each $o_i \in G_k$ do
13:                 $R \leftarrow R \cup (o_i, o_j, t_c)$
14:             continue
15:         $t_s \leftarrow \max\{t_c, t_{LB}\}$
16:         for each $o_i \in G_k$ do
17:             $triplet \leftarrow$ checkPair($o_i, o_j, t_s, t_f, \epsilon$)
18:             if $triplet \neq \emptyset$ then
19:                 $R \leftarrow R \cup \{triplet\}$
20: OIPT $\leftarrow$ OIPT $\cup \{(o_i, l_i, t_c) | o_i \in objBatch \wedge l_i = o_i.l\}$      ▷ Step 4
21: return $R$
```

pruning and probability-based pruning. Thus, the time complexity of Algorithm 3 is $O(|O'| \cdot |T_{FP}| \cdot m \cdot n)$, where $|O'| << |O|$ by our floor-based pruning. In addition, the time complexity of Algorithm 5 is $O\big((|objBatch|^2 + |objBatch| \cdot |O'|) \cdot (|T_{FP}| \cdot m \cdot n)\big)$, since Step 1 processes the object pairs within *objBatch*, and Step 3 processes the objects pairs (o_i, o_j), where $o_i \in objBatch$ and $o_j \in O'$.

Recall that we use the indexes $D2D$, $D2D_{id}$ and $P2P$ as introduced in Section 3.1. The total space complexity of the indexes is $O(2 \cdot |D|^2 + |P|^2)$, where the matrices $D2D$ and $D2D_{id}$ occupy $O(2 \cdot |D|^2)$, and the matrix $P2P$'s size is $O(|P|^2)$. Besides, each object occupies $O(m \cdot T_{Max})$, as we need to maintain the discrete samples in its uncertainty region and the corresponding probabilities at each timestamp through its lifespan.

5 EXPERIMENTAL STUDIES

We test our SDM framework with the batch processing algorithm **BP**. For comparison, we implement the following algorithms.

- **BPM**: The batch processing algorithm without the probability-based pruning (Lemma 4). To allow a fair comparison, we employ another time-parameterized distance-based pruning based on the maximum moving speed. In particular, given a timestamp t_p, the lower and upper bounds $dist(o_i, o_j, t)_{LB}$ and $dist(o_i, o_j, t)_{UB}$ for any later time $t \in (t_p, t_f]$ are obtained based on Equation 8.
- **QU**: The close contact for each object is processed one by one by calling the QueryUpdate algorithm (i.e., Algorithm 3).
- **QUM**: A QU variant whose probability-based pruning is replaced by the maximum moving speed based pruning used by BPM.

Note that there is no straightforward adaption of the solutions from [43] to our SDM problem, since (1) the nature of our problem requires a continuous solution; and (2) their index and solutions cannot handle the case that uncertainty regions expand with time.

All algorithms are implemented in Java and run on a Mac with a 2GHz Quad-Core Intel i5 CPU and 16GB memory.

Table 3: Parameter Settings

Parameter	Description	Settings
$\|O\|$	Object size	5k, **10k**, 15k, 20k, 25k, 30k
ϵ	Distance threshold	1, 2, **3**, 4, 5 (meters)
dia	Maximum diameter of $UR(o_i, tl_i)$	2, **4**, 6, 8, 10 (meters)
T_{Min}	Shortest update interval	3, **5**, 7, 9 (seconds)
T_{Max}	Longest update interval	10, 15, **20**, 25 (seconds)
T_{FP}	Future prediction interval	5, **10**, 15, 20 (seconds)
N_{floor}	Number of floors	10, **20**, 30

5.1 Results on Synthetic Data

5.1.1 Settings.

Indoor Space. Following [43], we use a floor plan based on a real world shopping mall. Each floor is 600m × 600m with 100 rooms, 4 hallways and 4 staircases. We obtain 141 partitions and 220 doors on each floor by decomposing those irregular hallways into smaller but regular ones. To generate a larger floor plan, we duplicate the floor 20 times, and obtain 2820 partitions and 4400 doors in total. The stairways are used to connect the two adjacent floors, each being 4m long. It takes less than 5 seconds to construct the indexes $D2D$, $D2D_{id}$ and $P2P$. In total, the indexes use approximately 8MB and are kept in the main memory.

Indoor Moving Objects. Following [24], we generate the indoor moving objects by the data generator Vita [22], where the objects' movements follow the random waypoint mobility model [16] with a maximum speed constraint $s_{max} = 1m/s$. First, we distribute $|O|$ objects evenly in the indoor space. Second, we gradually insert new objects as they enter the indoor space. In each second, the probability of we have some new objects entering is 1/4. The number of such entering objects follows the Poisson distribution with mean $\lambda = 1$. Third, we record the objects' exact locations every second. When an object updates, the corresponding location is used.

The probability pr of an object to update its location follows the Geometric distribution, where $pr = 1/(T_{Max} - T_{Min})$ for each second in $[T_{Min}, T_{Max}]$. An update replaces the old record in OIPT. The object is removed from the OIPT if it is not updated in a T_{Max} time interval since its last update. Nevertheless, the total number of objects in the OIPT remains roughly the same as time passes, as new objects are gradually inserted.

Following [43], we model each object o_i's uncertainty region at tl_i by a circle, with the diameter dia_i picked uniformly at random from $[1, dia]$. The pdf follows a Gaussian distribution with the circle center as the mean, and $(dia_i/6)^2$ as the variance. The initial uncertainty region is represented by ten sampling points, and when it enlarges as time passes, the number of sampling points increases accordingly to maintain the same level of sampling density.

Queries. We simulate the monitoring query for an hour, and the average processing time at each second is reported. For each setting, we run 5 times and report the average performance. Table 3 lists the parameter settings with default values in bold.

5.1.2 Efficiency Studies.

Effect of $|O|$. Figure 9 shows the results. As QU and QUM on $|O|$ = 30k take more than 2 seconds to run, the running times are not shown for better readability. The running times of all algorithms increase with an increasing $|O|$, as a larger $|O|$ leads to more candidate object pairs to be explored. Our BP runs faster than the competitors, and the gap becomes larger when $|O|$ increases, which shows the effectiveness of our pruning techniques and batch processing strategy. Moreover, BP (resp. QU) has a similar running time trend to BPM (resp. QUM), and we attribute their difference to the probability-based pruning described in Lemma 4.

Effect of *dia*. Figure 10 shows the results. The running times of all algorithms increase with an increasing *dia*, since an object with a larger *dia* is more likely to be involved in some candidate object pairs with other objects. BP always runs faster than the competitors. Moreover, BP (resp. QU) has a similar running time with BPM (resp. QUM) when *dia* is small (i.e., *dia* = 2), but runs much faster than BPM (resp. QUM) when *dia* is large. This could be explained by the fact that when *dia* increases, the area of the initial uncertainty region becomes larger, and thus the value of β is larger. Thus, the probability-based pruning is more effective and it reduces the number of candidate pairs that need to be explored.

Effect of ϵ. Figure 11 shows the results. The running times of the algorithms are insensitive to the changes of ϵ. It is because the search space of candidate objects from each object/group is dominated by the diameter of the uncertainty region (which is up to $T_{FP} \cdot s_{max}$), and thus increasing ϵ does not affect the running time. Moreover, BP always runs faster than the competitors.

Effect of T_{Min}. Figure 12 shows the results. The running times of BPM, QU and QUM decrease when T_{Min} increases. This is because a larger T_{Min} would lower the number of updates by the objects, and thus reducing the total number of objects processed. BP's running time is stable and BP still outperforms the competitors, as the pruning techniques enable BP to process the objects more efficiently.

Effect of T_{Max}. Figure 13 shows the results. The running times of the algorithms increase with an increasing T_{Max}, which is because the longer an object keeps alive, the more it has to compare with other objects, and thus increasing the running time. Nevertheless, BP runs consistently faster than its competitors, and it finishes within 0.2s when T_{Max} = 25.

Effect of T_{FP}. Figure 14 shows the results. As QUM on T_{FP} = 20 take more than 1.5 seconds to run, the running time is not shown for better readability. The running times of the algorithms increase with an increasing T_{FP}, and BP (resp. QU) runs much faster than BPM (resp. QUM). Since $T_{FP} \cdot s_{max}$ is the maximum possible indoor distance for which an object can traverse from its initial uncertainty region, a larger T_{FP} would lead to a looser distance lower bound $dist(o_i, o_j, t)_{LB}$, especially when the probability-based pruning is absent, and thus limiting the pruning effectiveness.

Effect of N_{floor}. We vary the number of floors while maintaining the same number of objects in the dataset. The results are shown in Figure 15. The running times of all algorithms decrease with an increasing number of floors. It is because when the number of floors increases, the average number of objects on each floor decreases. As mentioned in Section 4.1, staircases enable an effective way to reduce the number of candidate pairs, and distributing objects to more floors will further boost this pruning effect.

Effect of DDF. Besides Gaussian distribution, we run BP with 5 different distance decay functions [24]. The results are shown in Figure 16, where the full names of the DDF notations can be found in Table 4. According to the results, the running times of different DDFs are similar in general, while CL is usually the slowest. This is

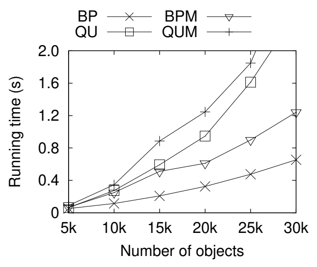


Figure 9: Effect of $|O|$.

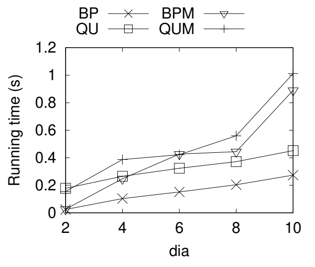


Figure 10: Effect of *dia*.

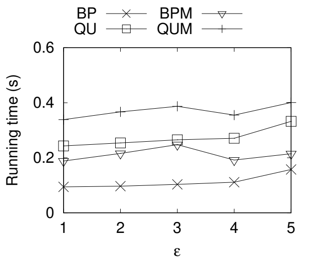


Figure 11: Effect of ϵ.

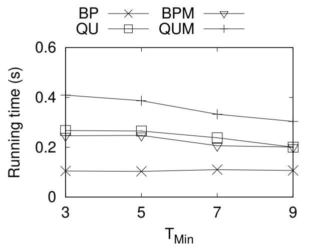


Figure 12: Effect of T_{Min}.

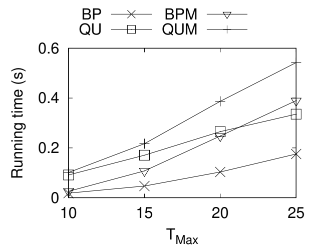


Figure 13: Effect of T_{Max}.

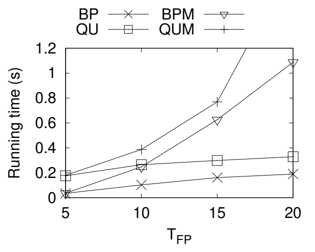


Figure 14: Effect of T_{FP}.

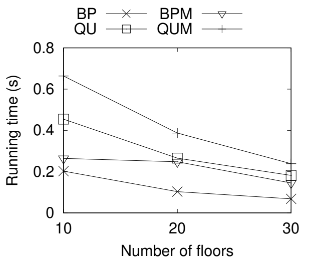


Figure 15: Effect of N_{floor}.

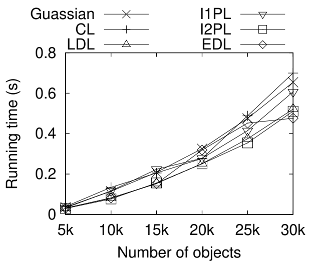


Figure 16: Effect of DDF.

Table 4: Precision of Output based on Different DDFs

	Precision		
	$w = 3$	$w = 5$	$w = 10$
Gaussian Distribution	0.4609	0.5683	0.6789
Constant Law (CL)	0.4622	0.5786	0.6972
Linear Decay Law (LDL)	0.5351	0.6508	0.7661
Inverse 1^{st} Power Law (I1PL)	0.4929	0.6049	0.7233
Inverse 2^{nd} Power Law (I2PL)	0.5274	0.6318	0.7500
Exponential Decay Law (EDL)	0.5243	0.6438	0.7606

because the probability-based pruning is the least effective under this function, since the β values decrease most rapidly over time.

5.1.3 Effectiveness Study. We evaluate the effectiveness of the proposed methods by comparing the prediction results with the actual expected distances. The synthetic dataset is used which contains the ground truth location for each object at each second. Thus, we can compute the actual expected distances from their ground truth locations. Specifically, we set $T_{Min} = 0$ and the probability pr of an object to update its location in each second to 1 in order to obtain their locations every second. Consider a timestamp t, we calculate the pairwise distances for all pairs of objects by their actual locations to find out those contacting pairs.

We evaluate the precision of the prediction. If the actual contact happens within w seconds after the predicted contact time, we count it as a hit. Note that our monitoring output has a recall of 1 as all actual contacts will be found at t_c. We run the experiment for 5 minutes using different DDFs.

The results for $w = \{3, 5, 10\}$ are shown in Table 4. According to the results, an average of around 50% of the pairs actually make the contacts within 3 seconds after the prediction, and around 70% make the contacts within 10 seconds. This means that our contact prediction algorithm is able to identify the future actual contacts.

5.2 Results on Real Data

Settings. We used a real Wi-Fi based positioning dataset [21], which contains positioning records in a shopping mall in Hangzhou, China on 1 Jan, 2018. The shopping mall has 7-floor, with size of around 108m × 80m, and contains ten staircases, where each is approximately 20m long. There are 977 partitions connected by 1613 doors. In total, the dataset contains 680,368 positioning records from 4,412 objects, spanning 24 hours. On average, an object updates its location every 15 seconds. Thus, we set $T_{Max} = 30$ seconds to take into account the possibility of missing an update.

We focus on BP here since it outperforms the competitors according to the previous experiments. We vary T_{FP} in $\{5, 10, 15, 20\}$.
Effect of *dia*. Figure 17 shows the results. The running times of all settings increase with an increasing *dia*, as more object pairs need to be processed. The running time also increases when T_{FP} increases. It is because a larger T_{FP} corresponds to a longer prediction interval, and thus the number of candidate pairs at each timestamp is increased accordingly.
Effect of ϵ. Figure 18 shows the results. The running times of all settings are insensitive to ϵ. This is because, when we search candidate object pairs for an object or a group, only a small portion of the search space is contributed by ϵ, while a much larger portion comes from the diameter of the uncertainty region.
Effect of T_{Min}. Figure 19 shows the results. The running times of all settings decrease when T_{Min} increases. The number of objects updated in each second is smaller with a larger T_{Min}, and fewer object pairs need to be processed.
Effect of T_{Max}. Figure 20 shows the results. The running times increase as T_{Max} increases. This is because, as the objects stay alive for a longer time, more candidate object pairs need to be processed, thus increasing the processing time.

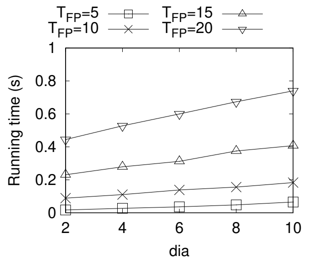


Figure 17: Effect of *dia*.

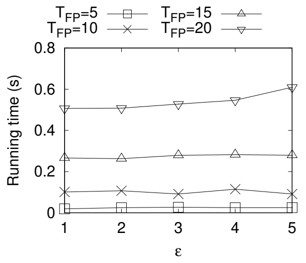


Figure 18: Effect of ϵ.

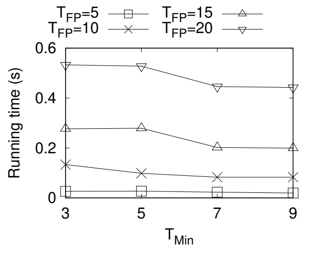


Figure 19: Effect of T_{Min}.

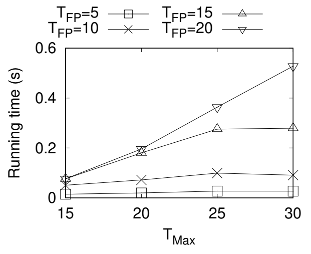


Figure 20: Effect of T_{Max}.

Table 5: Existing Indoor Continuous Queries

	Object	Continuous Query	Distance -aware
Probabilistic *k*NN [49]	Moving (Online)	-	√
Range and *k*NN [51]	Moving (Online)	-	√
Spatio-temporal Join [30]	Moving (Historical)	-	-
Top-*k* Popular Location [23]	Moving (Historical)	-	-
Top-*k* Dense Region [24]	Moving (Online)	-	-
Distance-aware Join [42, 43]	Moving (Online)	-	√
Continuous Detour [38]	Static	√	-
Continuous Range [52]	Static	√	√
Continuous Range [48]	Moving (Online)	√	-
SDM (this work)	Moving (Online)	√	√

6 RELATED WORK

Querying Moving Objects in Indoor Space. Lu et al. [29] designed an indoor space model and an indexing framework to facilitate indoor shortest path finding. Shao et al. [39] proposed the VIP-Tree and KP-Tree that enable efficient processing of indoor shortest path queries and spatial keyword queries. Liu et al. [28] proposed an indoor crowd model, and proposed the indoor crowd-aware path queries that find the routes that are the fastest or least crowded. Xie et al. [42, 43] studied the distance-aware join for indoor moving objects. They designed a composite index scheme and algorithms to answer the indoor range join query and *k*NN query. Li et al. [24] studied the problems of finding the top-*k* popular locations from uncertain historical indoor positioning data. The authors also studied searching for online indoor dense regions [23], where the object location uncertainty is integrated into the definitions for computing the density of the indoor regions. Yang et al. [49] proposed the probabilistic threshold *k*NN query in indoor space. Yu et al. [51] improved over previous works by proposing a particle filter based method. Lu et al. [30] studied the spatial-temporal joins on symbolic indoor tracking data. Although these works also studied indoor moving objects, they are *snapshot queries*, i.e., these queries cannot be used for continuously monitoring and the algorithms do not involve dynamic result updates, and thus are not applicable to our problem.

Salgado et al. [38] studied the continuous detour query in indoor space. Yang et al. [48] studied the continuous monitoring of moving objects in the symbolic indoor space and proposed an infrastructure for indoor range monitoring. Yuan and Schneider [52] studied the continuous range query in an indoor space. None of these works consider pairwise distances between objects that are essential for our social distance monitoring problem. We summarize and compare all existing queries and our SDM in Table 5.

Continuous Query in Outdoor Space. The continuous intersection join over moving objects was studied in [54, 55]. Tang et al. [40] studied the distributed online tracking problem. Their focus is to minimize the communication cost between the tracker (system) and the observers (objects), which is orthogonal to our problem that focuses on the efficient computation in the system. Xu et al. [45, 46] formulated the continuous range query over the multi-attribute trajectories. A large number of works solve the continuous query problems by the safe region techniques [6, 7, 44]. A comprehensive survey could be found in [36]. Hu et al. [13] aimed at settling the location update issue and devised a common interface to monitor mixed types of continuous spatial queries. Wang et al. [41] investigated the problem of continuous spatial-keyword query over streaming data. Chow et al. [10] designed a continuous answer maintenance scheme to maintain a query answer.

Social Distancing and Contact Tracing. Chao et al. [5] studied the trajectory contact search query for the contact tracing problem. The query and algorithms are based on historical trajectories, and thus are not applicable to our problem. Kato et al. [17] proposed a trajectory-based private contact tracing system that checks whether the user visits the infected location. Xu et al. [47] proposed a toolbox called IMO for simulating and querying the infected query objects. Some studies [1, 35, 50] measure and monitor the social distance among people by analyzing surveillance videos using machine learning and deep learning approaches.

7 CONCLUSION

In this paper, we studied continuous indoor social distance monitoring (SDM). SDM monitors and predicts the distances between object pairs, and finds those pairs that will be in close contact soon. We proposed a framework for SDM in an online setting, and developed efficient algorithms to update the results. Extensive experiments were conducted on both real and synthetic datasets. The results verified the efficiency and scalability of our proposals.

For future work, we can take the environment of each room into account to derive a tailor-made distance threshold for each room. For example, the threshold for a room without ventilation should be set to much smaller than a room with air purifiers. It is also interesting to extend the framework to provide alternative route suggestions to users who want to avoid contact with others.

ACKNOWLEDGMENTS

This work was supported by Independent Research Fund Denmark (No. 8022-00366B). Hua Lu is the corresponding author.

REFERENCES

[1] Imran Ahmed, Misbah Ahmad, Joel JPC Rodrigues, Gwanggil Jeon, and Sadia Din. 2021. A deep learning-based social distance monitoring framework for COVID-19. *Sustainable Cities and Society* 65 (2021), 102571.

[2] Oguzhan Alagoz, Ajay K Sethi, Brian W Patterson, Matthew Churpek, and Nasia Safdar. 2021. Effect of timing of and adherence to social distancing measures on COVID-19 Burden in the United States: A Simulation Modeling Approach. *Annals of internal medicine* 174, 1 (2021), 50–57.

[3] Akinori Asahara, Kishiko Maruyama, Akiko Sato, and Kouichi Seto. 2011. Pedestrian-movement prediction based on mixed Markov-chain model. In *ACM SIGSPATIAL*. 25–33.

[4] Artur Baniukevic, Dovydas Sabonis, Christian S Jensen, and Hua Lu. 2011. Improving wi-fi based indoor positioning using bluetooth add-ons. In *MDM*. IEEE, 246–255.

[5] Pingfu Chao, Dan He, Lei Li, Mengxuan Zhang, and Xiaofang Zhou. 2021. Efficient Trajectory Contact Query Processing. In *DASFAA*. Springer, 658–666.

[6] Muhammad Aamir Cheema, Ljiljana Brankovic, Xuemin Lin, Wenjie Zhang, and Wei Wang. 2010. Multi-guarded safe zone: An effective technique to monitor moving circular range queries. In *ICDE*. IEEE, 189–200.

[7] Muhammad Aamir Cheema, Xuemin Lin, Ying Zhang, Wei Wang, and Wenjie Zhang. 2009. Lazy updates: An efficient technique to continuously monitoring reverse knn. *Proceedings of the VLDB Endowment* 2, 1 (2009), 1138–1149.

[8] Reynold Cheng, Dmitri V Kalashnikov, and Sunil Prabhakar. 2003. Evaluating probabilistic queries over imprecise data. In *SIGMOD*. 551–562.

[9] Reynold Cheng, Dmitri V Kalashnikov, and Sunil Prabhakar. 2004. Querying imprecise data in moving object environments. *TKDE* 9 (2004), 1112–1127.

[10] Chi-Yin Chow, Mohamed F Mokbel, and Hong Va Leong. 2011. On efficient and scalable support of continuous queries in mobile peer-to-peer environments. *IEEE Transactions on Mobile Computing* 10, 10 (2011), 1473–1487.

[11] Pavel Davidson, Jussi Collin, and Jarmo Takala. 2010. Application of particle filters for indoor positioning using floor plans. In *2010 Ubiquitous Positioning Indoor Navigation and Location Based Service*. IEEE, 1–4.

[12] Fredrik Gustafsson, Fredrik Gunnarsson, Niclas Bergman, Urban Forssell, Jonas Jansson, Rickard Karlsson, and P-J Nordlund. 2002. Particle filters for positioning, navigation, and tracking. *IEEE Transactions on signal processing* 50, 2 (2002), 425–437.

[13] Haibo Hu, Jianliang Xu, and Dik Lun Lee. 2005. A generic framework for monitoring continuous spatial queries over moving objects. In *SIGMOD*. 479–490.

[14] Peggy L Jenkins, Thomas J Phillips, Elliot J Mulberg, and Steve P Hui. 1992. Activity patterns of Californians: use of and proximity to indoor pollutant sources. *Atmospheric Environment. Part A. General Topics* 26, 12 (1992), 2141–2148.

[15] Christian S Jensen, Dan Lin, and Beng Chin Ooi. 2004. Query and update efficient B+-tree based indexing of moving objects. In *VLDB*, Vol. 30. VLDB Endowment, 768–779.

[16] David B Johnson and David A Maltz. 1996. Dynamic source routing in ad hoc wireless networks. In *Mobile computing*. Springer, 153–181.

[17] Fumiyuki Kato, Yang Cao, and Masatoshi Yoshikawa. 2020. Secure and Efficient Trajectory-Based Contact Tracing using Trusted Hardware. In *2020 IEEE International Conference on Big Data (Big Data)*. IEEE, 4016–4025.

[18] Neil E Klepeis, William C Nelson, Wayne R Ott, John P Robinson, Andy M Tsang, Paul Switzer, Joseph V Behar, Stephen C Hern, and William H Engelmann. 2001. The National Human Activity Pattern Survey (NHAPS): a resource for assessing exposure to environmental pollutants. *Journal of Exposure Science & Environmental Epidemiology* 11, 3 (2001), 231–252.

[19] Jayakanth Kunhoth, AbdelGhani Karkar, Somaya Al-Maadeed, and Abdulla Al-Ali. 2020. Indoor positioning and wayfinding systems: a survey. *Human-centric Computing and Information Sciences* 10, 1 (2020), 1–41.

[20] Sohee Kwon, Amit D Joshi, Chun-Han Lo, David A Drew, Long H Nguyen, Chuan-Guo Guo, Wenjie Ma, Raaj S Mehta, Erica T Warner, Christina M Astley, et al. 2020. Association of social distancing and masking with risk of COVID-19. *medRxiv* (2020).

[21] Huan Li, Hua Lu, Gang Chen, Ke Chen, Qinkuang Chen, and Lidan Shou. 2020. Toward translating raw indoor positioning data into mobility semantics. *ACM Transactions on Data Science* 1, 4 (2020), 1–37.

[22] Huan Li, Hua Lu, Xin Chen, Gang Chen, Ke Chen, and Lidan Shou. 2016. Vita: A versatile toolkit for generating indoor mobility data for real-world buildings. *Proceedings of the VLDB Endowment* 9, 13 (2016), 1453–1456.

[23] Huan Li, Hua Lu, Lidan Shou, Gang Chen, and Ke Chen. 2018. Finding most popular indoor semantic locations using uncertain mobility data. *TKDE* 31, 11 (2018), 2108–2123.

[24] Huan Li, Hua Lu, Lidan Shou, Gang Chen, and Ke Chen. 2018. In search of indoor dense regions: An approach using indoor positioning data. *TKDE* 30, 8 (2018), 1481–1495.

[25] Xiang Lian and Lei Chen. 2008. Monochromatic and bichromatic reverse skyline search over uncertain databases. In *SIGMOD*. 213–226.

[26] Xiang Lian and Lei Chen. 2008. Probabilistic group nearest neighbor queries in uncertain databases. *TKDE* 20, 6 (2008), 809–824.

[27] Xiang Lian and Lei Chen. 2010. Similarity join processing on uncertain data streams. *TKDE* 23, 11 (2010), 1718–1734.

[28] Tiantian Liu, Huan Li, Hua Lu, Muhammad Aamir Cheema, and Lidan Shou. 2021. Towards crowd-aware indoor path planning. *Proceedings of the VLDB Endowment* 14, 8 (2021), 1365–1377.

[29] Hua Lu, Xin Cao, and Christian S Jensen. 2012. A foundation for efficient indoor distance-aware query processing. In *ICDE*. IEEE, 438–449.

[30] Hua Lu, Bin Yang, and Christian S Jensen. 2011. Spatio-temporal joins on symbolic indoor tracking data. In *ICDE*. IEEE, 816–827.

[31] Wesley Mathew, Ruben Raposo, and Bruno Martins. 2012. Predicting future locations with hidden Markov models. In *Proceedings of the 2012 ACM conference on ubiquitous computing*. 911–918.

[32] Jignesh M Patel, Yun Chen, and V Prasad Chakka. 2004. STRIPES: an efficient index for predicted trajectories. In *SIGMOD*. 635–646.

[33] Jan Petzold, Andreas Pietzowski, Faruk Bagci, Wolfgang Trumler, and Theo Ungerer. 2005. Prediction of indoor movements using bayesian networks. In *International Symposium on Location-and Context-Awareness*. Springer, 211–222.

[34] Pratap S Prasad and Prathima Agrawal. 2010. Movement prediction in wireless networks using mobility traces. In *2010 7th IEEE Consumer Communications and Networking Conference*. IEEE, 1–5.

[35] Narinder Singh Punn, Sanjay Kumar Sonbhadra, Sonali Agarwal, and Gaurav Rai. 2020. Monitoring COVID-19 social distancing with person detection and tracking via fine-tuned YOLO v3 and Deepsort techniques. *arXiv preprint arXiv:2005.01385* (2020).

[36] Jianzhong Qi, Rui Zhang, Christian S Jensen, Kotagiri Ramamohanarao, and Jiayuan He. 2018. Continuous spatial query processing: A survey of safe region based techniques. *ACM Computing Surveys (CSUR)* 51, 3 (2018), 1–39.

[37] Shaojie Qiao, Dayong Shen, Xiaoteng Wang, Nan Han, and William Zhu. 2014. A self-adaptive parameter selection trajectory prediction approach via hidden Markov models. *IEEE Transactions on Intelligent Transportation Systems* 16, 1 (2014), 284–296.

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

[39] Zhou Shao, Muhammad Aamir Cheema, David Taniar, Hua Lu, and Shiyu Yang. 2020. Efficiently Processing Spatial and Keyword Queries in Indoor Venues. *TKDE* (2020).

[40] Mingwang Tang, Feifei Li, and Yufei Tao. 2015. Distributed online tracking. In *SIGMOD*. 2047–2061.

[41] Xiang Wang, Ying Zhang, Wenjie Zhang, Xuemin Lin, and Wei Wang. 2015. Ap-tree: Efficiently support continuous spatial-keyword queries over stream. In *ICDE*. IEEE, 1107–1118.

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

[43] Xike Xie, Hua Lu, and Torben Bach Pedersen. 2014. Distance-aware join for indoor moving objects. *TKDE* 27, 2 (2014), 428–442.

[44] Hongfei Xu, Yu Gu, Yu Sun, Jianzhong Qi, Ge Yu, and Rui Zhang. 2020. Efficient processing of moving collective spatial keyword queries. *The VLDB Journal* 29, 4 (2020), 841–865.

[45] Jianqiu Xu, Zhifeng Bao, and Hua Lu. 2019. Continuous range queries over multi-attribute trajectories. In *ICDE*. IEEE, 1610–1613.

[46] Jianqiu Xu, Zhifeng Bao, and Hua Lu. 2021. A Framework to Support Continuous Range Queries over Multi-Attribute Trajectories. *TKDE* (2021).

[47] Jianqiu Xu, Hua Lu, and Zhifeng Bao. 2020. IMO: a toolbox for simulating and querying" infected" moving objects. *Proceedings of the VLDB Endowment* 13, 12 (2020), 2825–2828.

[48] Bin Yang, Hua Lu, and Christian S Jensen. 2009. Scalable continuous range monitoring of moving objects in symbolic indoor space. In *ACM CIKM*. 671–680.

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

[50] Dongfang Yang, Ekim Yurtsever, Vishnu Renganathan, Keith A Redmill, and Ümit Özgüner. 2021. A vision-based social distancing and critical density detection system for COVID-19. *Sensors* 21, 13 (2021), 4608.

[51] Jiao Yu, Wei-Shinn Ku, Min-Te Sun, and Hua Lu. 2013. An RFID and particle filter-based indoor spatial query evaluation system. In *EDBT*. 263–274.

[52] Wenjie Yuan and Markus Schneider. 2010. Supporting continuous range queries in indoor space. In *MDM*. IEEE, 209–214.

[53] Jun Zhang, Dimitris Papadias, Kyriakos Mouratidis, and Manli Zhu. 2004. Spatial queries in the presence of obstacles. In *EDBT*. Springer, 366–384.

[54] Rui Zhang, Dan Lin, Kotagiri Ramamohanarao, and Elisa Bertino. 2008. Continuous intersection joins over moving objects. In *ICDE*. IEEE, 863–872.

[55] Rui Zhang, Jianzhong Qi, Dan Lin, Wei Wang, and Raymond Chi-Wing Wong. 2012. A highly optimized algorithm for continuous intersection join queries over moving objects. *The VLDB Journal* 21, 4 (2012), 561–586.