Reachability Queries with Transfer Decay

Elena V. Strzheletska
Department of Computer Science and Engineering,
University of California, Riverside,
elenas@cs.ucr.edu

Vassilis J. Tsotras
Department of Computer Science and Engineering,
University of California, Riverside,
tsotras@cs.ucr.edu

***Abstract*—A spatiotemporal reachability query identifies whether a physical item (or information, virus etc.) could have been transferred from the source moving object O_S to the target moving object O_T during a time interval I (either directly, or through a chain of intermediate transfers). Previous work on spatiotemporal reachability queries, assumes the transferred information remains the same. This paper introduces a novel reachability query under the scenario of *information decay*. Such queries arise when the value of information (virus load etc.) that travels through the chain of intermediate objects *decreases* with each transfer. To address such queries efficiently over large spatiotemporal datasets, we introduce the RICCdecay algorithm. An experimental evaluation shows the efficiency of the proposed algorithm over previous approaches.**

***Index Terms*—spatio-temporal data, reachability queries.**

I. Introduction

Answering reachability queries on large spatiotemporal datasets is important for a wide range of applications, such as security monitoring, surveillance, public health, epidemiology, social networks, etc. Nowadays, with the perpetuation of Covid-19, the reachability and trajectory analysis are as important as ever, since efficient contact tracing helps to control the spread of the disease.

Given two objects O_S and O_T, and a time interval I, a spatiotemporal reachability query identifies whether information (or physical item etc.) could have been transferred from O_S to O_T during I (typically indirectly through a chain of intermediate transfers). The time to exchange information (or physical items etc.) between objects affects the problem solution and it is application specific. An 'instant exchange' scenario (where information can be instantly transferred and retransmitted between objects) is assumed in [1], but may not be the case in many real world applications. On the other hand, [2] and [3] consider two reachability scenarios without the 'instant exchange' assumption: reachability with *processing delay* and *transfer delay*. After two objects encountered each other, the contacted object may have to spend some time to process the received information before being able to exchange it again (processing delay). In other applications, for the transfer of information to occur, two objects are required to stay close to each other for some period of time (transfer delay); we call such elongated contact a *meeting*. To contract the virus, one has to be exposed to an infected person for a brief period of time; to exchange messages through Bluetooth, two cars have to travel closely together for some time.

The problems discussed above had a common feature: the value of information carried by the object that initiated the information transmission process and the value of information obtained by any reached object was assumed to remain unchanged. In this paper, we remove this assumption, since for some applications it may not be valid. For example, if two persons communicate over the phone (or a Bluetooth-enabled device), some information may be lost due to faulty connection. We introduce a *reachability with transfer decay* problem, where the value of the transmitted item experiences a *decay* with each transfer. Note that we will still assume the transfer delay scenario as this is more realistic.

In this paper, we present algorithm RICCdecay, that can efficiently compute reachability with transfer decay queries on large spatiotemporal datasets. More details and query extensions within this decay framework appear in [4]. The rest of the paper is structured as follows: Section II is an overview of related work, Section III defines the problem. Sections IV and V describe the preprocessing and query processing of RICCdecay. Section VI contains the experimental evaluation and Section VII concludes the paper.

II. Related Work

Graph Reachability. A large number of works is proposed for the static graph reachability problem. They are categorized in [5] as those, that use: (i) transitive closure compression [6], [7], (ii) hop labeling [8], [9], and (iii) refined online search [10], [11]. In our model, the reachability question can be represented as a variation of a shortest path query. The state-of-the-art algorithm for solving shortest path problems on road networks is Contraction Hierarchies (CH) [12].

Evolving Graphs. In [13], an external hierarchical index structure is used for efficient storing and retrieving of historical graph snapshots. For large dynamic graphs, [14] constructs a reachability index, based on labeling, ordering, and updating techniques.

Spatiotemporal Databases. A survey on spatiotemporal access methods appears in [15]. Such indexes often involve some variation on hierarchical trees [16]–[18], some form of a grid-based structure [19], or indexing in parametric space [20], [21]. The existing spatiotemporal indexes support traditional range and nearest neighbor queries Recently complex queries have focused on identifying the behavior and patterns of moving objects (i.e., flocks, ROIs, clusters) [22]–[25].

Spatiotemporal Reachability Queries. The first disk-based solutions for the spatiotemporal reachability problem, ReachGrid and ReachGraph appeared in [1]. These are indexes on the contact datasets that enable faster query times. ReachGraph makes the assumption that a contact between two objects can be instantaneous, which allows it to be smaller in size and thus reduces query time. ReachGrid does not have this assumption.

In [2], two novel types of the 'no instant exchange' spatiotemporal reachability queries were introduced: reachability queries with *processing* and *transfer delays (meetings)*. The proposed solution to the first type utilized CH [12] for *path contraction*. Later, [3] considered the second type of delays and introduced two algorithms, RICCmeetMin and RICCmeetMax. To reduce query processing time, these algorithms precompute the shortest valid (RICCmeetMin), and the longest possible meetings (RICCmeetMax) respectively. Neither one of them can accommodate reachability queries with decay.

III. Problem Description

A. Background

Let $O = \{O_1, O_2, ..., O_n\}$ be a set of moving objects, whose locations are recorded for a long period of time at discrete time instants $t_1, t_2, ..., t_i, ...$, with the time interval between consecutive location recordings $\Delta t = t_{k+1} - t_k$ ($k = 1, 2, ...$) being constant. A *trajectory* of a moving object O_i is a sequence of pairs (l_i, t_k), where l_i is the location of object O_i at time t_k. Two objects, O_i and O_j, that at time t_k are respectively at positions l_i and l_j, have a *contact* (denoted as $< O_i, O_j, t_k >$), if $dist(l_i, l_j) \leq d_{cont}$, where d_{cont} is the *contact distance* (a distance threshold given by the application), and $dist(l_i, l_j)$ is the Euclidean distance between the locations of objects O_i and O_j at time t_k.

The reachability with transfer delay scenario requires to discretize each $[t_k,\ t_{k+1})$ by dividing it into a series of non-overlapping subintervals of equal duration $\Delta\tau = \tau_{i+1} - \tau_i$, such that $\tau_0 = t_k$ and $\tau_r = t_{k+1}$. Two objects, O_i and O_j, had a *meeting* $< O_i, O_j, I_m >$ during $I_m = [\tau_s, \tau_f]$ if they had been within d_{cont} from each other at each $\tau_k \in [\tau_s, \tau_f]$. The *duration* of this meeting is $m = \tau_f - \tau_s$. A meeting is *valid* if $m \geq m_q\Delta\tau$ (where m_q is the query specifies *required meeting duration* - time, needed for the objects to complete the exchange). Object O_T is *(m_q)-reachable* from object O_S during time interval $I = [\tau'_s, \tau'_f]$, if there exists a chain of subsequent valid meetings $< O_S, O_{i_1}, I_{m_0} >$, $< O_{i_1}, O_{i_2}, I_{m_1} >$, ... ,$< O_{i_k}, O_T, I_{m_k} >$, where each $I_{m_j} = [\tau_{s_j}, \tau_{f_j}]$ is such that $\tau_{f_j} - \tau_{s_j} \geq m_q$, $\tau'_s \leq \tau_{s_0}$, $\tau_{f_k} \leq \tau'_f$, and $\tau_{s_{j+1}} \geq \tau_{f_j}$ for $j = 0, 1, ..., k-1$. A reachability query determines whether the target object O_T is reachable from the source object O_S during time interval I.

Consider the example in Fig. 1. Table (a) shows the actual meetings between all objects during one time block, which is given as a meetings graph in (b). A materialized reachability graph shows how the information is being dispersed. Suppose, O_1 is the source object and $m_q = 2\Delta\tau$. Then graph G_2 in (c) is the materialized (m_q)-reachability graph for O_1 on data from (a). By looking at G_2, one can discover all objects that can be (m_q)-reached by O_1 during the time interval $[\tau_0, \tau_8]$.

B. Reachability with Decay

In the reachability with transfer delay scenario, to complete the transfer, it is necessary for the objects to stay within the contact distance for at least m_q time units. Under some circumstances, the transfer may still fail to occur, or the value of the transferred item may go down. We consider a new type of reachability scenario, namely *reachability with transfer decay*, that accounts for such events.

Let d denote the *rate of transfer decay* - a part of information lost during one transfer ($d \in [0, 1)$). Then $p = 1 - d$ ($p \in (0, 1]$) will define the portion of the transfered information. Suppose, the weight of the item carried by a source object O_S is w. Then, during a valid meeting, O_S can transfer this item to some object O_i. However, considering the decay, if $d > 0$, the value of information, obtained by O_i lessens and becomes wp. With each further transfer, the value of the received item will continue to decrease. This process can be modeled with an exponential decay function.

We denote the number of transfers (hops), that is required to pass the information from object O_S to object O_i as h ($h \geq 0$). If O_i cannot be reached by O_S, $h = \infty$. Let $g_w : \mathbb{R} \to \mathbb{R}$ be a function that calculates the weight of an item after h transfers. Assuming that the transfer decay d and thus p are constant for the same item, $g_w(h)$ can be defined as follows:

$$g_w(h) = wp^h. \tag{1}$$

The number of transfers h in (1) depends on the time when it is being evaluated, and denoted as $h(O_i^{\tau_j})$. Consider Fig. 1: O_1 can reach O_3 by $\tau = 6$ with 3 hops, while it requires only 1 hop to reach O_3 by $\tau = 8$. So, $h(O_3^{\tau_6})$ = 3 and $h(O_3^{\tau_8})$ = 1.

The case with $p = 1$ corresponds to the reachability with transfer delay problem [3]. If $p < 1$, the value of $g_w(h)$ decreases exponentially with each transfer. Let ν denote the *threshold weight*. If after some transfer, the weight of the item becomes smaller than the threshold weight ν, we disregard that event by assigning to the newly transferred item the weight of 0. We say, that h is the *allowed number of hops (transfers)* if it satisfies the threshold weight inequality

$$g_w(h) \geq \nu. \tag{2}$$

We denote the *maximum allowed number of transfers* that satisfies inequality (2) as h_{max}. Let $f_w : \mathbb{R} \to \mathbb{R}$ be a function that assigns the weight to an item carried by object O_i at time τ_j, and denote it as $f_w(O_i^{(\tau_j)})$. (For brevity, we say 'the weight of object O_i at time τ_j'.) We define $f_w(O_i^{(\tau_j)})$ as follows:

Object 1	Object 2	Meeting	
O_1	O_4	0	2
O_1	O_3	6	8
O_2	O_3	4	6
O_2	O_4	2	4

(a)

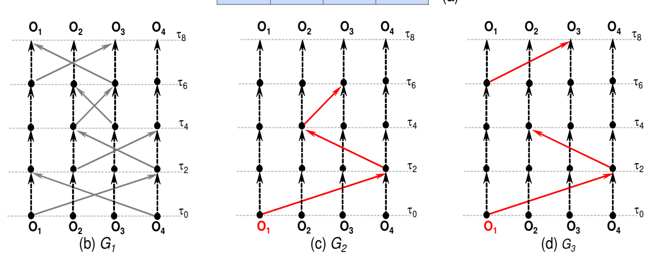

Fig. 1: (a) Record of meetings; (b) G_1-meetings graph; (c) G_2-materialized reachability with 'transfer delay' graph; (d) G_3-materialized reachability with 'transfer decay' graph; (source object O_1, $m_q = 2\Delta\tau$, $d = 0.2$, $\nu = 0.6$, $I = [\tau_0, \tau_8]$).

Time	Weight function / Object	O_1	O_2	O_3	O_4
τ_0	g_w	1	0	0	0
	f_{w_1}	1	0	0	0
	f_{w_2}	1	0	0	0
τ_2	g_w	1	0	0	0.8
	f_{w_1}	1	0	0	0.8
	f_{w_2}	1	0	0	0.8
τ_4	g_w	1	0.64	0	0.8
	f_{w_1}	1	0.64	0	0.8
	f_{w_2}	1	0	0	0.8
τ_6	g_w	1	0.64	0.512	0.8
	f_{w_1}	1	0.64	0	0.8
	f_{w_2}	1	0	0	0.8
τ_8	g_w	1	0.64	0.8	0.8
	f_{w_1}	1	0.64	0.8	0.8
	f_{w_2}	1	0	0.8	0.8

Fig. 2: The actual weight of an item g_w and its assigned weights f_{w_1} and f_{w_2}, calculated on data from Table 1(a) (source object $O_1, p = 0.8$, $\nu = 0.6$ for f_{w_1} and $\nu = 0.7$ for f_{w_2}).

$$f_w(O_i^{(\tau_j)}) = \begin{cases} g_w(h) & \text{if } h(O_i^{(\tau_j)}) \leq h_{max}, \\ 0 & \text{otherwise.} \end{cases} \quad (3)$$

The table in Fig. 1(a) shows the meetings between objects O_1, O_2, O_3, and O_4 during the time interval $I = [\tau_0; \tau_8]$. Here we assume again that O_1 is the source object, $m_q = 2\Delta\tau$ and $d = 0.2$ (thus $p = 0.8$). To illustrate the difference between the actual weight of an item g_w and its assigned weight f_w, the values g_w, f_{w_1}, and f_{w_2} are computed for each object at time instants from τ_0 to τ_8 and recorded in the table (see Fig. 2). The values for the assigned weight functions f_{w_1} and f_{w_2} are computed for $\nu = 0.6$ and $\nu = 0.7$ respectively. The graph G_3 in Figure 1(d) is constructed for f_{w_1}.

Object O_T is *(m_q, d)-reachable* from object O_S during time interval $I = [\tau'_s, \tau'_f]$, if there exists a chain of subsequent valid and successful (under m_q, d conditions) meetings $< O_S, O_{i_1}, I_{m_0} >$, $< O_{i_1}, O_{i_2}, I_{m_1} >$, ... ,$< O_{i_k}, O_T, I_{m_k} >$, where each $I_{m_j} = [\tau_{s_j}, \tau_{f_j}]$ is such that, $\tau'_s \leq \tau_{s_0}$, $\tau_{f_k} \leq \tau'_f$, and $\tau_{s_{j+1}} \geq \tau_{f_j}$ for $j = 0, 1, ..., k-1$. The earliest time when O_T can be reached is denoted as $\tau_R(O_T)$.

We assume that the values of d and ν are query specified. An *(m_q, d)-reachability* query Q_{md}: $\{O_S, O_T, w, d, I, m_q, \nu\}$ determines whether the target object O_T is reachable from the source object O_S, that caries an item whose weight is w, during time interval $I = [\tau_s, \tau_f]$, given required meeting duration m_q, rate of transfer decay d, and threshold weight ν, and reports the earliest time instant when O_T was reached.

IV. Preprocessing

In order to simplify the presentation, we assume that the minimum meeting duration μ ($\mu \leq m_q$) is known before the preprocessing, and set m_q = μ, thus fixing it. However, the proposed algorithm can be extended to work with any query specified m_q by combining it with RICCmeetMax [3].

Suppose, our datasets contain records of objects' locations ordered by the location reporting time t. We start by dividing the time domain into non-overlapping time intervals of equal duration - *time blocks* (denoted as B_k). Each B_k contains all records whose reporting times belong to the corresponding time period. The number of the reporting times in each block is the *contraction parameter* C, which is discussed in Section VI.

Next, for each B_k, the following steps have to be completed: (i) computing candidate contacts, (ii) verifying contacts (at each t_k), (iii) identifying meetings, (iv) computing reachability, and (v) index construction. Steps (i), (ii), (iii), and (v) are similar to those in [3]; we discuss them briefly, while concentrating on the most challenging step *(iv)*.

Information regarding each object O_i is saved in a data structure *objectRecord*(O_i), which is created at the beginning of B_k and deleted at the end, after all the needed information is written on the disk. *ObjectRecord*(O_i) contains *Object_id*, *Cell_id* (the object's placement in the grid with side H when it was first seen during B_k), $ContactsRec$ and $MeetingsRec$ (records of the contacts and meetings of O_i during B_k). The grid side H is a parameter, which is discussed in Section VI.

A. Computing Contacts and Finding Meetings

Two objects O_i and O_j are *candidate contacts* at time t_k if the distance between them is no greater than $d_{cc} = 2d_{max} + d_{cont}$ at t_k (where d_{max} is the largest distance that can be covered by any object during Δt). Candidate contacts can potentially have a contact between t_k and t_{k+1}. At each t_k we partition the area covered by the dataset into cells with side d_{cc}. Now all candidate contacts of O_i are in the same with O_i or neighboring cells.

Assuming that between t_k and t_{k+1} objects move linearly, at t_{k+1}, we can verify if there were indeed any contacts between each pair of candidate contacts during $[t_k, t_{k+1})$. If a contact occurred, it is saved in the list $ContactsRec$ of $objectRecord$ of each contacted object. If an object O_i had O_j for its contact at two or more consecutive time instants, these contacts are merged into a meeting, and written in the $MeetingsRec$ list. At the end of each B_k, m is computed for each meeting. All meetings with $m < \mu$ (with the exception of boundary meetings) are pruned, while all the remaining meetings are recorded into file *Meetings*. Boundary meetings are recorded regardless of their duration since they may span more than one block, which needs to be verified during the query processing.

B. Computing Reachability

To speed up the query time, for each object O_i, we precompute all objects that are (μ, d)-reachable from O_i during B_k. However, to find, which objects can be (μ, d)-reached by O_i, we need to know d and ν, which are assumed to be unknown at the preprocessing time. To overcome this issue, we turn our problem into a *hop-reachability* problem.

One of the requirements for object O_T to be reachable from object O_S is that each meeting in the chain of meetings from O_S to O_T has to be a *successful* meeting. It follows from (2), that after each meeting, for each companion object O_i, the following condition must hold: $g_w(h) = wp^h \geq \nu$.

Thus, $h \leq \log_p \frac{\nu}{w}$, and finally

$$h_{max} = \lfloor \log_p \frac{\nu}{w} \rfloor. \quad (4)$$

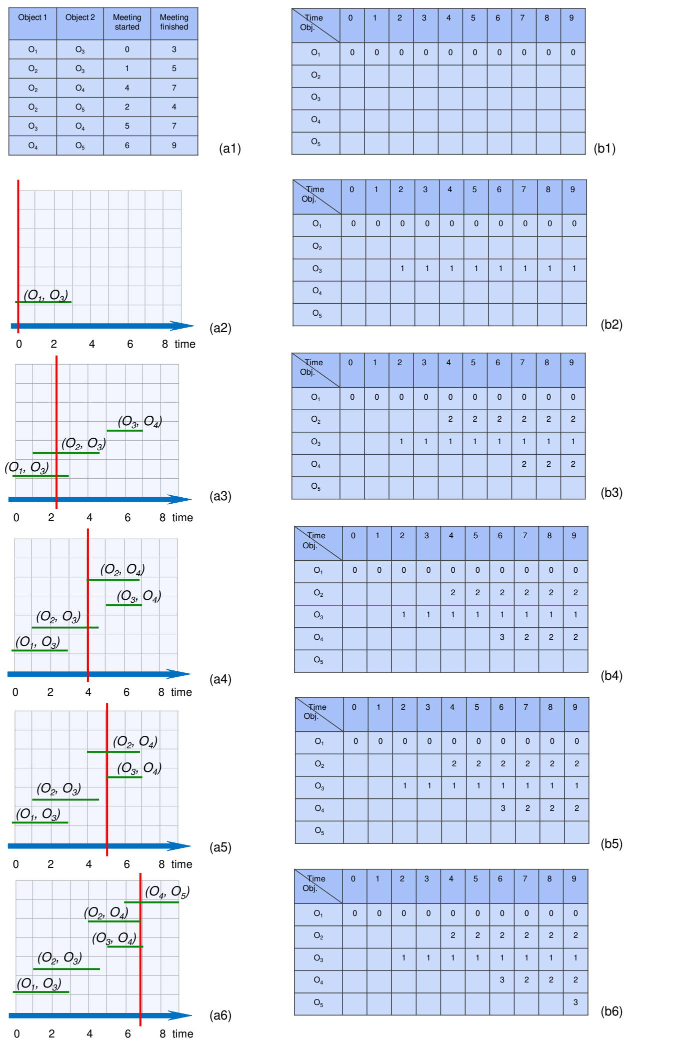

Fig. 3: Computing (h_{min})-reachable objects from O_1 ($\mu = 2$).

Now the problem can be stated as follows: for each object O_i, compute all objects, that are (μ, h_{max})-reachable from O_i. Moreover, for each object O_j reached by O_i, we find the minimum number of such transfers $h_{min} \leq h_{max}$.

Our algorithm makes use of plane sweep algorithm. Consider the data in the table (a1) of Fig. 3. It contains records of actual meetings between all objects during one time block. (a2)-(a6) describe how reached objects and meetings are being discovered. The information about the 'reachability' status of each object is recorded into a temporary table, which is created at the beginning of each block. A row is added to the table for each reached object at the time when it is reached, and it is updated with any new event. The development of the reachability table is shown in (b1)-(b6).

We show how to compute all objects reached by object O_1 during the given time block, assuming that $\mu = 2\Delta\tau$. At the beginning of the block, the sweep line is positioned at $\tau = 0$, and only O_1 is reached (with $h_{min} = 0$), which is recorded in (b1). During the given time block, O_1 has only one meeting, $< O_1, O_3, [0, 3] >$ which is placed on the plane (a2). As a result of this meeting, O_3 is reached at time $\tau = 2$, with $h_{min} = 1$, which is recorded in (b2). The sweep line moves to the time $\tau = 2$ - time, when O_3 was reached. Next, all meetings of O_3 that are either active at $\tau = 2$ or start after this time, are materialized. These are meetings $< O_3, O_2, [1, 5] >$ and $< O_3, O_4, [5, 7] >$. Consider the first meeting: $< O_3, O_2, [1, 5] >$. It begins at $\tau = 1$, but the retransmission does not start until $\tau = 2$, since only at this time O_3 becomes reached. So O_2 and O_4 become reached at $\tau = 4$ and $\tau = 7$ respectively, with $h_{min} = 2$ ((a3), (b3)). The line changes its position to $\tau = 4$. This process continues until the sweep line reaches the end of the time block. Note that the earliest reached time for an object may change, also an object's h_{min} value may decrease with time. For example, object O_4 was reached by O_2 with $h_{min} = 3$ at $\tau = 6$ ((a4), (b4)), however as a result of the meeting with object O_3, its h_{min} value went down to $h_{min} = 2$ at $\tau = 7$ ((a3), (b3)).

Algorithm 1 Reach(h_{min})

1: Input: O_S
2: procedure UpdateHmin (O_i, τ_s, τ_f, h)
3: for each $\tau_k \in [\tau_s, \tau_f]$ do $h_{min}(O_i^{\tau_k}) = h$
4: **for** each O_i **do**
5: $\tau_R(O_i) = \infty$
6: UpdateHmin($O_i, \tau_0, \tau_{end}, \infty$) ▷ τ_0 and τ_{end} are the first and last time units of a block
7: **procedure** REACHHOP(O_S)
8: $time = 0$, $\tau_R(O_S) = 0$, UpdateHmin($O_S, \tau_0, \tau_{end}, 0$), $S_{PQ} = \{O_S\}$, $S_{ReachHop} = \{\emptyset\}$
9: **while** ((S_{PQ}) $\neq \{\emptyset\}$ and time $\leq \tau_{end}$) **do**
10: $O_i = ExtractMin\ (S_{PQ})$
11: $S_{ReachHop} = S_{ReachHop} \cup O_i$, $time = \tau_R\ (O_i)$
12: **for** each O_j that had a valid meeting with O_i **do**
13: **if** $O_j \notin S_{ReachHop}$ **then**
14: $\tau_{Rnew}(O_j) = \infty$
15: **while** $\tau_{Rnew}(O_j) \geq \tau_R(O_j)$ **do**
16: read next meeting $M_{ij} =< O_i, O_j, [\tau_s, \tau_f] >$
17: compute $\tau_{Rnew}(O_j)$
18: **if** $\tau_{Rnew}(O_j) < \tau_R(O_j)$ **then**
19: *Update* (S_{PQ}, O_j), $h = h_{min}(O_i^{time}) + 1$
20: **if** $\tau_R(O_j) = \infty$ **then** $\tau_R(O_j) = \tau_{end+1}$
21: UpdateHmin($O_j, \tau_{Rnew}, \tau_R(O_j) - 1, h$))
22: **if** ($M_{ij} = last\ meeting < O_i, O_j >$ in B_k) **then**
23: $\tau_{Rnew}(O_j) = -1$
24: **return** $S_{Reached}$

The process for computing all objects that are (h_{min})-reachable by O_S during one time block is generalized in Algorithm 1. Procedure UpdateHmin initializes and then updates the table that records the reachability status of each reached object. The $S_{ReachHop}$ set keeps all objects for which all h_{min} values as well as the earliest reached time had been computed and finalized. Those objects that were found to be reached, but not in $S_{ReachHop}$ yet, are placed in the priority queue S_{PQ}, where priority to the objects is given according to their 'reached' times. When an object (say object O_i) that has the earliest reached time ($\tau_R(O_i)$) is extracted from S_{PQ}, it is placed into $S_{ReachHop}$ (lines 10, 11). At this time, all meetings of objects that can be reached by O_i (but not in $S_{ReachHop}$) are analyzed (lines 13 - 23). As a result, $\tau_R(O_j)$ and h_{min} may change (lines 19 and 21).

C. Index Construction

The index structure of RICCdecay is similar to the one of RICCmeet algorithms [3]: to enable an efficient search in the files *Meetings* and *Reached(Hop)* during the query processing, we create three index files: *Meetings Index*, *Reached Index*, and *Time Block Index* (Fig. 4).

V. Query Processing

The reachability with decay query Q_{md} is issued in the form Q_{md}: $\{O_S, O_T, w, d, [\tau_s, \tau_f], \mu, \nu\}$. (Recall that during the preprocessing, for simplicity, we set $m_q = \mu$.) First, using equation (4), we rewrite the problem as hop-reachability problem, replacing w, d, and ν from Q_{md} with h_{max}. The new query can be written as Q_{mh}: $\{O_S, O_T, h_{max}, [\tau_s, \tau_f], \mu\}$.

The processing of Q_{mh} starts from computing the time blocks B_s, ... , B_f that contain data for the query interval $I = [\tau_s, \tau_f]$. File *Time Block Index* (accessed only once per query) points to the pages in the *Meetings Index* and *Reached Index* that correspond to the required blocks. These index files (accessed once per time block) in turn point to the appropriate pages in files *Meetings* and *Reached(Hop)* respectively.

The set of reached objects $S'_{reached}$ is initialized with object O_S at the beginning of the query processing. We start reading file *Reached(Hop)* from block B_s, retrieving all records for O_S. Durin B_k, an object O_j cannot be considered as reached unless $h_{min}(O_j^{B_k}) \leq h_{max}$ (where $h_{min}(O_j^{B_k})$ is the value h_{min} of object O_j at the end of B_k). So, each objects O_j that was found to be reached by O_S, is added to $S'_{reached}$, along with h_{min}, provided that $h_{min}(O_j^{B_s}) \leq h_{max}$. Next, we proceed to block B_{s+1}. This time, retrieving all the companions of each object from $S'_{reached}$ and updating it by either adding new objects or adjusting the h_{min} value for the objects that are already in the set. Such adjustment may be needed if, for some object $O_i \in S'_{reached}$, $h_{min}(O_j^{B_s}) > h_{min}(O_j^{B_{s+1}})$. The process continues until O_T is added to $S'_{reached}$ while reading some block $B_i (i < f)$ or the last block B_f is reached.

If at the end of processing B_f, $S'_{reached}$ does not contain the target O_T, the query processing can be aborted, otherwise it moves to the file *Meetings*. The process of identifying reached objects inside each block is the same as the one described in Algorithm 1. If there is a meeting between O_i and O_j that ends at the end of the time block, but is shorter than m_q, we check if it continues in the next block, and merge two meetings into

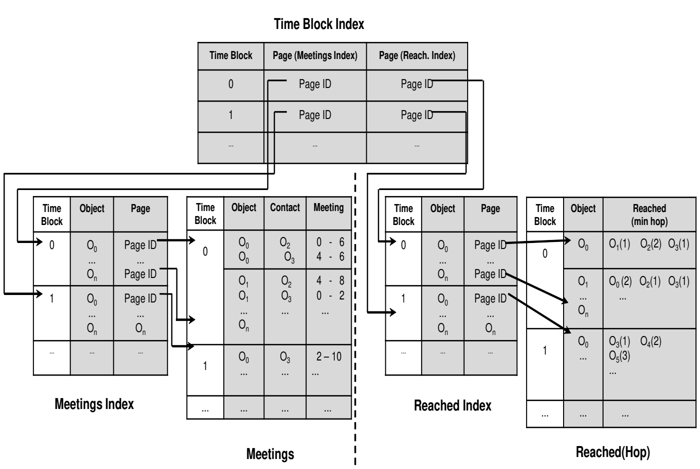

Fig. 4: Two-level index on files *Meetings* and *Reached(Hop)*.

one if needed. Also, if during B_k object O_i was reached by O_S with $h_{min}(O_i^{B_k}) = h_1$, and in a later block B_m, O_j was reached by O_i within h_2 hops, $h_{min}(O_j^{B_m}) = h_1 + h_2$. Object O_j is considered to be reached by O_S if $h_{min}(O_j^{B_m}) \leq h_{max}$.

If by the end of B_i, O_T was not found to be reached, and $B_i < B_f$, the search switches to *Reached(Hop)*. This process continues until O_T is confirmed to be reached by the information from *Meetings*, or the last block B_f is processed.

VI. Experimental Evaluation

We proceed with the results of the experimental evaluation of RICCdecay. Since there are no other algorithms for processing spatiotemporal reachability queries with decay, we compare RICCdecay against a modified version of RICCmeetMin [3] that enables it to answer such queries. All experiments were performed on a Linux system with a 3.4GHz Intel CPU, 16 GB RAM, 3TB disk and 4K page size. All programs were written on C++ and compiled using gcc version 4.8.5 with optimization level 3.

TABLE I: Size of datasets, auxiliary files and indexes

Dataset		MV_1	MV_2	MV_4	RW_1	RW_2	RW_4
Size of Dataset (GB)		54	107	213	97	194	387
Auxiliary Data and Index Size (GB)	RICCmeetMin	4.6	23	83.3	11.6	44.9	157
	RICCdecay	5.2	27.7	98	12.7	50	178.7

All experiments were performed on six realistic datasets of two types: Moving Vehicles (MV) and Random Walk (RW). The MV datasets were created by the Brinkhoff data generator [26], and contain information about $1000, 2000$, and 4000 vehicles respectively (denoted as MV_1, MV_2, and MV_4). We set $d_{cont} = 100$ m (for a (class 1) Bluetooth connection).

For the RW datasets, we created our own generator (see details in [2], [3]). RW datasets consist of trajectories of $10000, 20000$, and 40000 individuals respectively (denoted as RW_1, RW_2, and RW_4). The size of each dataset (in GB) appears in Table I. We set $d_{cont} = 10$ m, to identify physical contacts or contacts in the range of Bluetooth-enabled devices.

The performance was evaluated in terms of disk I/Os during query processing. The ratio of a sequential I/O to a random I/O is system dependent; for our experiments this ratio is 20:1 (20 sequential I/Os take the same time as 1 random). We thus present the equivalent number of random I/Os using this ratio.

A. Parameter Optimization

The values of the contraction parameter C and the grid resolution H, that are used for the preprocessing, were tuned on the 5% subset of each dataset as follows. We performed the preprocessing of each subset for different values of (C, H), and tested the performance of RICCdecay on a set of 200 queries, varying the query length between 500 and 3500 sec for the MV, and between 600 and 4200 sec for the RW datasets.

The h_{max} value was picked uniformly at random from 1 to 4 (we stopped at $h_{max} = 4$ since the higher the h_{max}, the less information is caried by the reached object and thus presents less interest). The parameters C and H were varied as follows: H - from 500 to 40000 m for MV, and from 250 to 2000 m for RW datasets; and C - from 0.5 to 30 min. For each dataset, the pair (C, H) that minimized the number of I/Os was used for the rest of the experiments. For example, for MV_1 we used $C = 14$ min and $H = 20000$ m, while for RW_4 we used $C = 2$ min nd $H = 500$ m.

B. Preprocessing Space and Time

The sizes of the auxiliary files and the index sizes for RICCmeetMin and RICCdecay appear in Table I.

C. Query Processing

The performance of RICCdecay was tested on sets of 100 queries of different time intervals and various $h_{max} = 1, 2, 3, 4$, while μ was set to 2 sec, and the initial weight w of the item carried by O_S was set to 1 for all the experiments.

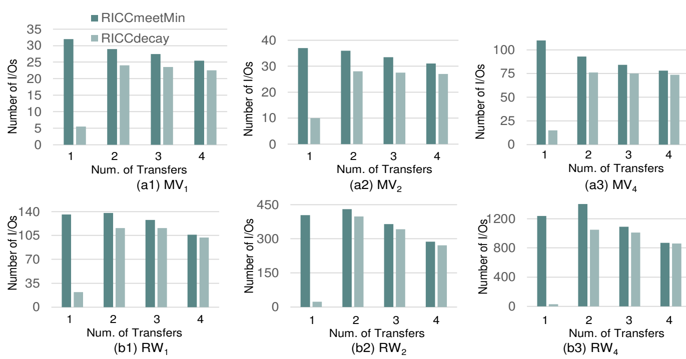

Fig. 5: Increasing maximum allowed number of transfers

Increasing the Maximum Allowed Number of Transfers. First, we analyze the impact of h_{max} on the performance of the RICCdecay. We ran a set of 100 queries varying h_{max} from 1 to 4; each query's interval was picked uniformly at random from 500 to 3500 sec for the MV datasets, and from 600 to 4200 sec for RW datasets. The results are presented in Fig. 5 $(a1-b3)$. RICCdecay accesses from 1.8 (for $MV2$ dataset) to 11.5 (for $RW4$ dataset) times less pages than RICCmeetMin.

Increasing Query Length. Now we test the performance of RICCdecay for various query lengths and compare with that of RICCmeetMin. Each test was run on a set of 100 queries varying query length from 500 to 3500 sec for MV, and from 600 to 4200 sec for RW datasets. The h_{max} value for each query was picked uniformly at random from 1 to 4. The results are shown in Fig. 6. For these sets of queries, RICCdecay outperforms RICCmeetMin in all the tests, accessing about 44% less pages in average, and this result does not change significantly from one dataset to another.

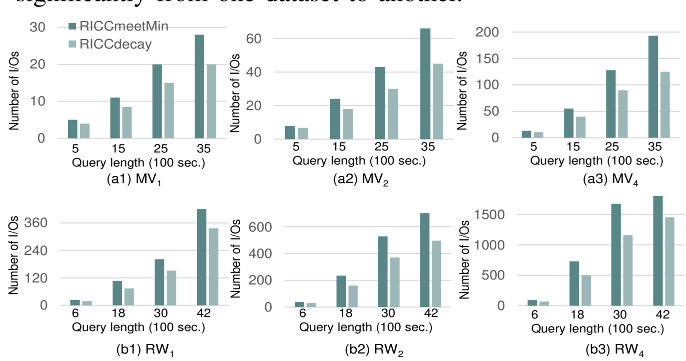

Fig. 6: Increasing query length

VII. Conclusions

We presented a novel reachability problem on reachability with transfer decay. To process these queries efficiently, we designed algorithm RICCdecay and tested it on six realistic datasets against a modified version of the RICCmeetMin algorithm [3]. The performance comparison showed that RICCdecay is more efficient on the new types of queries.

Acknowledgements: This work was partially supported by NSF grants SES-1831615 and IIS-1901379.

References

[1] H. Shirani-Mehr, F. B. Kashani, and C. Shahabi, "Efficient reachability query evaluation in large spatiotemporal contact datasets," *PVLDB*, vol. 5, no. 9, 2012.

[2] E. V. Strzheletska and V. J. Tsotras, "RICC: fast reachability query processing on large spatiotemporal datasets," in *SSTD*, 2015, pp. 3–21.

[3] ——, "Efficient processing of reachability queries with meetings," in *Proceedings of the 25th ACM SIGSPATIAL*, 2017, pp. 22:1–22:10.

[4] ——, "Reachability and Top-k Reachability Queries with Transfer Decay," *arXiv preprint arXiv:2105.08312*, 2021.

[5] E. Jin, N. Ruan, S. Dey, and J. Y. Xu, "Scarab: scaling reachability computation on large graphs," in *ACM SIGMOD*, 2012, pp. 169–180.

[6] R. Agrawal, A.Borgida, and H.V.Jagadish, "Efficient Managemet on Transitive Relationships in Large Data and Knowledge Bases," in *ACM SIGMOD*, 1989, pp. 253–262.

[7] H. Wang, H. He, J. Yang, P. S. Yu, and J. X. Yu, "Dual labeling: Answering graph reachability queries in constant time," in *IEEE ICDE*, 2006, pp. 75–75.

[8] E. Cohen, E. Halperin, H. Kaplan, and U. Zwick, "Reachability and distance queries via 2-hop labels," *SIAM Journal on Computing*, vol. 32, no. 5, pp. 1338–1355, 2003.

[9] J. Cai and C. K. Poon, "Path-hop: efficiently indexing large graphs for reachability queries," in *ACM CIKM*, 2010, pp. 119–128.

[10] H. Yildirim, V. Chaoji, and M. J. Zaki, "GRAIL: scalable reachability index for large graphs," in *PVLDB*, 2010, pp. 276–284.

[11] F. Merz and P. Sanders, "PReaCH: A Fast Lightweight Reachability Index Using Pruning and Contraction Hierarchies," in *ESA Symp.*, 2014, pp. 701–712.

[12] R. Geisberger, P. Sanders, D. Schultes, and D. Delling, "Contraction hierarchies: faster and simpler hierarchical routing in road networks," in *7th Intl. Conf. on Experimental algorithms*, 2008, pp. 319–333.

[13] U. Khurana and A. Deshpande, "Efficient snapshot retrieval over historical graph data," in *IEEE ICDE*, 2013, pp. 997–1008.

[14] A. D. Zhu, W. Lin, S. Wang, and X. Xiao, "Reachability queries on large dynamic graphs: a total order approach," in *ACM SIGMOD*, 2014, pp. 1323–1334.

[15] L. Nguyen-Dinh, W. G. Aref, and M. F. Mokbel, "Spatio-temporal Access Methods: Part2 (2003 - 2010)," *IEEE Data Engineering Bulletin*, vol. 33, no. 2, pp. 46–55, 2010.

[16] G. Kollios, D. Gunopulos, and V. Tsotras, "On indexing mobile objects," in *ACM PODS*, 1999, pp. 261–272.

[17] D. Pfoser, C. S. Jensen, and Y. Theodoridis, "Novel approaches in query processing for moving object trajectories," in *VLDB*, 2000, pp. 395–406.

[18] M. Hadjieleftheriou, G. Kollios, V. J. Tsotras, and D. Gunopulos, "Efficient indexing of spatiotemporal objects," in *EDBT*, 2002, pp. 251–268.

[19] X. Xiong, M. F. Mokbel, and W. G. Aref, "Lugrid: Update-tolerant grid-based indexing for moving objects," in *MDM*, 2006, p. 13.

[20] J. Ni and C. Ravishankar, "Indexing spatiotemporal trajectories with efficient polynomial approximation," *IEEE TKDE*, vol. 19, no. 5, 2007.

[21] P. Bakalov, M. Hadjieleftheriou, E. Keogh, and V. Tsotras, "Efficient trajectory joins using symbolic representations," *MDM*, pp. 86–93, 2005.

[22] C. Jensen, D. Lin, and B. Ooi, "Continuous clustering of moving objects," *IEEE TKDE*, vol. 19, no. 9, pp. 1161–1174, 2007.

[23] P. Kalnis, N. Mamoulis, and S. Bakiras, "On discovering moving clusters in spatio-temporal data," in *SSTD*, 2005, pp. 364–381.

[24] M. R. Vieira, P. Bakalov, and V.J.Tsotras, "On-line discovery of flock patterns in spatio-temporal data," in *ACM GIS*, 2009, pp. 286–295.

[25] M. R. Uddin, C. Ravishankar, and V. J. Tsotras, "Finding regions of interest from trajectory data," in *MDM*, 2011, pp. 39–48.

[26] T. Brinkhoff *et al.*, "Generating traffic data," *IEEE Data Eng. Bull.*, vol. 26, no. 2, pp. 19–25, 2003.