Reachability and Top-k 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—The prevalence of location tracking systems has resulted in large volumes of spatiotemporal data generated every day. Addressing reachability queries on such datasets is important for a wide range of applications (surveillance, public health, social networks, etc.). A spatiotemporal reachability query identifies whether a physical item (or information etc.) could have been transferred from the source object $O_S$ to the target object $O_T$ during a time interval $I$ (either directly, or through a chain of intermediate transfers). In previous research on spatiotemporal reachability queries, the number of such transfers is not limited, and the weight of a piece of transferred information remains the same. This paper introduces novel reachability queries, which assume a scenario of information decay. Such queries arise when the value of information 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. Further, the decay scenario leads to an important extension: if there are many different sources of information, the aggregate value of information an object can obtain varies. As a result, we introduce a top-k reachability problem, identifying the $k$ objects with the highest accumulated information. We also present the RICCTopK algorithm that can efficiently compute top-k reachability with transfer decay queries. An experimental evaluation shows the efficiency of the proposed algorithms over previous approaches.

Index Terms—spatio-temporal data, reachability query, top-k, access methods.

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.

While the problems discussed above considered different reachability scenarios, they 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 name a reachability problem, where the value of the transmitted item experiences a decay with each transfer, as reachability with transfer decay. The formal definition of the new problem is given in Section III. Note that in this paper will still assume the transfer delay scenario as this is more realistic.

The information decay scenario leads to the second problem we introduce, namely top-k reachability with decay. Consider a group of objects (people, cars, etc...), each of which possesses a different piece of information, and starts its transmission to other objects independently of each other. The objects that initiated the process form a set of source objects. Each of the source objects may carry information of a different value (and different weight), and during a contact, a decay of each piece of information may not be the same. As time progresses, any object may receive one or more items that originally came from different sources. It is reasonable to compute the combined weight of all the items collected by each object and rank the objects according to their aggregate weights. Objects with the most aggregated information may be of special interest. A top-k reachability query with decay finds the $k$ objects with the highest aggregate weights.

In this paper we present two algorithms: RICCdecay and RICCTopK, that can efficiently compute reachability and top-k...
reachability with transfer decay queries on large spatiotemporal datasets. RICCdecay consists of two stages, preprocessing and query processing, while RICCTopK performs top-k query processing using the index from RICCdecay. The rest of the paper is structured as follows: Section II is an overview of related work while Section III defines the two problems. Section IV describes the RICCdecay algorithm and its preprocessing phase, while Sections V and VI present the query processing for the reachability with decay and the top-k reachability problems respectively. Section VII contains the experimental evaluation and Section VIII concludes the paper.

II. RELATED WORK

Graph Reachability. A large number of works is proposed for the static graph reachability problem. The efficient approaches balance the preprocessing with the query processing, and are categorized in [4] as those, that use: (i) transitive closure compression [5], [6], (ii) hop labeling [7], [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]. CH benefits from creating a hierarchy of nodes on the basis of their importance for the given road network. In our problem, there is no node preference between the graph nodes, and thus applying for it CH would be inefficient.

Evolving Graphs. Evolving graphs have recently received increased attention The DeltaGraph [13] is an external hierarchical index structure used for efficient storing and retrieving of historical graph snapshots. For analyzing distance and reachability on temporal graphs, [14] utilizes graph reachability labeling, while for large dynamic graphs, [15] constructs a reachability index, based on a combination of labeling, ordering, and updating techniques. These methods work with datasets of a different nature, compared with spatiotemporal.

Spatiotemporal Databases. A survey on spatiotemporal access methods appears on in [16]. They often involve some variation on hierarchical trees [17]–[21], some form of a grid-based structure [22], [23], or indexing in parametric space [24], [25]. The existing spatiotemporal indexes support traditional range and nearest neighbor queries and not the reachability queries we examine here. Some of the recent more complex queries were focused on querying/identifying the behavior and patterns of moving objects: discovering moving clusters [26], [27], flock patterns [28], and convoy queries [29].

Spatiotemporal Reachability Queries. The first disk-based solutions for the spatiotemporal reachability problem, ReachGrid and ReachGraph appeared in [11]. These are indexes on the contact datasets that enable faster query times. In ReachGrid, during query processing only a necessary portion of the contact network is constructed and traversed. In ReachGraph, the reachability at different scales is precomputed and then reused at query time. ReachGraph makes the assumption that a contact between two objects can be instantaneous, and thus during one time instance, a chain of contacts may occur, which allows it to be smaller in size and thus reduces query time. ReachGrid does not require the ‘instant exchange’ 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.

Spatiotemporal Top-k Queries. While many works have considered variations of spatial and spatiotemporal top-k queries [30]–[36], no previous work addresses the decay scenario.

III. PROBLEM DESCRIPTION

In this section, we define two novel spatiotemporal reachability problems: the problem of reachability with decay and its extension, the problem of top-k reachability with decay.

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 $t_1, t_2, ..., t_i, ..., \text{...}$, 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 $\text{dist}(l_i, l_j) \leq d\text{\_cont}$, where $d\text{\_cont}$ is the contact distance (a distance threshold given by the application), and $\text{dist}(l_i, l_j)$ is the Euclidean distance between the locations of objects $O_i$ and $O_j$ at time $t_k$.

The reachability with transfer delay scenario (which we follow here) requires to discretize the time interval between consecutive position readings $[t_k, t_{k+1}]$ by dividing it into a series of non-overlapping subintervals $[\tau_0, \tau_1), ..., [\tau_i, \tau_{i+1}), ..., [\tau_{r-1}, \tau_r)$ of equal duration $\Delta \tau = \tau_{i+1} - \tau_i$, such that $\tau_0 = k$ and $\tau_r = t_{k+1}$. We say that two objects, $O_i$ and $O_j$, had a meeting $< O_i, O_j, I_m >$ during the time interval $I_m = [\tau_s, \tau_f]$ if they had been within the threshold distance $d\text{\_cont}$ from each other at each time instant $\tau_k \in [\tau_s, \tau_f]$. The duration of this meeting is $m = \tau_f - \tau_s$. We call a meeting valid if its duration $m \geq m_q \Delta \tau$ (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 $< O_S, O_{i_1}, I_{m_0} >$, $< O_{i_1}, O_{i_2}, I_{m_1} >$, ..., $< O_{i_{k-1}}, O_T, I_{m_k} >$, where each $I_{m_j} = [\tau_{s_j}, \tau_{f_j}]$ is such that $\tau_{f_j} - \tau_{s_j} \geq m_q \Delta \tau$, $\tau_{s_j} \leq \tau_{s_0}$, $\tau_{f_j} \leq \tau_f'$, and $\tau_{s_{j+1}} \geq \tau_{f_j}$, for $j = 0, 1, ..., k-1$. A reachability query determines whether object $O_T$ (the target) is reachable from object $O_S$ (the source) during time interval $I$.

Consider example in Fig. I 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 object $O_1$ is the source object and the required meeting duration $m_q = 2\Delta \tau$. 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 object \(O_1\) during the time interval \(I = [t_0, t_8]\).

\[
\text{Object 1, Object 2, Meeting} \\
\begin{array}{ccc}
O_1 & O_2 & 0 \\
O_2 & O_1 & 2 \\
O_1 & O_3 & 8 \\
O_2 & O_3 & 8 \\
O_3 & O_1 & 2 \\
O_3 & O_2 & 8 \\
\end{array}
\]

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 delay’ graph; (source object \(O_1\), \(m_q = 2\Delta\tau\), \(d = 0.2\), \(\nu = 0.6\), \(I = [t_0, t_8]\)).

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 a time interval that is at least as long as the required meeting duration \(m_q\). However, even if a meeting between objects \(O_i\) and \(O_j\) satisfied the \(m_q\) requirement, under some circumstances, the transfer may still fail to occur, or the value of the transferred item may go down (e.g., a complete or partial signal loss during the communication). 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 transferred 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 \(g_w : \mathbb{R} \rightarrow \mathbb{R}\) 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 equation (1), that an item has to complete in order to be delivered from object \(O_S\) to object \(O_i\), depends on the time \(\tau_j\) when it is being evaluated, and thus denoted as \(h(O_{ij}^\tau)\). Consider example in Fig. 1 Suppose again that \(m_q = 2\Delta\tau\) and object \(O_1\) is the source object. It can reach object \(O_3\) by \(\tau = 6\) with 3 hops, while it requires only one hop for object \(O_1\) to reach \(O_3\) by \(\tau = 8\). So, \(h(O_{31}^\tau) = 3\) and \(h(O_{31}^\tau) = 1\).

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 \(\nu\) denote the threshold weight. If after some transfer, the weight of the item becomes smaller than the threshold weight \(\nu\), 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 \(h_{\text{max}}\). Let \(f_w : \mathbb{R} \rightarrow \mathbb{R}\) be a function that assigns the weight to an item carried by object \(O_i\) at time \(\tau_j\), and denote it as \(f_w(O_{ij}(\tau_j))\). (For brevity, we say ‘the weight of object \(O_i\) at time \(\tau_j\)’. We define \(f_w(O_{ij}(\tau_j))\) as follows:

\[
f_w(O_{ij}(\tau_j)) = \begin{cases} 
g_w(h) & \text{if } h(O_{ij}(\tau_j)) \leq h_{\text{max}}, \\
0 & \text{otherwise.} \end{cases} \tag{3}
\]

The table in Fig. 1(a) shows the meetings between objects \(O_1, O_2, O_3\), and \(O_4\) during the time interval \(I = [t_0, t_8]\). 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 \(t_0\) to \(t_8\) 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 \(< O_S, O_{i_1}, I_{m_1} >, < O_{i_1}, O_{i_2}, I_{m_2} >, \ldots, < O_{i_k}, O_T, I_{m_k} >\), where each \(I_{m_j} = [\tau_{s_j}, \tau_{f_j}]\) is such that, \(\tau_{s_j} \leq \tau_{s_{j+1}}\), \(\tau_{f_j} \leq \tau_{f_{j+1}}\), and \(\tau_{s_{j+1}} \geq \tau_{f_j}\) for \(j = 0, 1, \ldots, k-1\). The earliest time when \(O_T\) can be reached is denoted as \(\tau_R(O_T)\).
We assume that the values of \(d\) and \(\nu\) are query specified. An \((m_q, d)\)-reachability query \(Q_{md} : \{O_S, OT, w, d, I, m_q, v\}\) determines whether the target object \(O_T\) is reachable from the source object \(O_S\), that carries 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 \(\nu\), and reports the earliest time instant when \(O_T\) was reached.

### C. Top-k Reachability with Decay

We now consider the problem of top-k reachability with transfer decay. Let \(S = \{O_{S_1}, O_{S_2}, \ldots, O_{S_n}\}\), \(W = \{w_1, w_2, \ldots, w_q\}\), and \(D = \{d_1, d_2, \ldots, d_q\}\) be the sets of source objects, weights, and decays respectively. Each object \(O_{S_i} \in S\) carries a different piece of information (or physical item), whose weight is \(w_{r_i}\), and is able to transfer this information following the \((m_q, d)\)-reachability scenario. The transfer decay for the item carried by object \(O_{S_i}\) is \(d_{r_i}\).

As the objects move through the network, source objects \(O_{S_i}\) encounter other objects, and may pass information to them. Since each source object owns a different piece of information, the transferred weight depends on both, the number of hops and the source that it came from. Let \(h_r (h_r \geq 0)\) be the number of hops required for object \(O_{S_i}\) to pass the information to \(O_i\). Then we can calculate the actual weight of an item \(r\) after \(h_r\) transfers using equation (1) as:

\[
g_{w(r)}(h_r) = w_{r_i}p_{i}^{h_r},
\]

where \(r = (1, 2, \ldots, q)\). As in the previous problem, we require that each threshold weight inequality has been satisfied:

\[
g_{w(r)}(h_r) \geq \nu
\]

for \(r = (1, 2, \ldots, q)\) and threshold weight \(\nu\).

Let \(h_{\text{max}}(r)\) be the maximum allowed number of transfers that satisfies the inequality above for each \(r = (1, 2, \ldots, q)\). Similarly to (3), function \(f_{w(r)}\) assigns weight to the \(r^{th}\) item carried by object \(O_i\) at time \(\tau_j\) (denoted as \(f_{w(r)}(O_i^{(\tau_j)})\)). We define the assigned weight \(f_{w(r)}(O_i^{(\tau_j)})\) as:

\[
f_{w(r)}(O_i^{(\tau_j)}) = \begin{cases} g_{w(r)}(h_r) & \text{if } h_r(O_i^{(\tau_j)}) \leq h_{\text{max}}(r), \\ 0 & \text{otherwise}. \end{cases}
\]

Furthermore, each object may receive more than one item. We denote the aggregate weight function \(F_w : \mathbb{R} \rightarrow \mathbb{R}\) that assigns weight to the collection of items carried by object \(O_i\) at time \(\tau_j\) as \(F_w(O_i^{(\tau_j)})\), and define it as follows:

\[
F_w(O_i^{(\tau_j)}) = \sum_{r=1}^{q} f_{w(r)}(O_i^{(\tau_j)}),
\]

where each \(f_{w(r)}(O_i^{(\tau_j)})\) is computed as in (4).

A top-k reachability with decay query \(Q_{topK}\) is given in the form \(\{S, W, D, I, m_q, \nu, k\}\). The goal of \(Q_{topK}\) is to find \(k\) objects with the highest aggregate weight \(F_w\) (computed according to (5)), that was obtained during the time interval \(I\).

### IV. Preprocessing

As with other reachability problems discussed above, there are two naive approaches to solve \((m_q, d)\)-reachability problem: (i) ‘no-preprocessing’, and (ii) ‘precompute all’. Neither one of them is feasible for large graphs: the first does not involve any preprocessing, and thus too slow during the query processing, while the second requires too much time for preprocessing and too much space for storing the preprocessed data. To overcome the disadvantages of the second approach and still achieve fast query processing, we precompute and store only some data as described below.

In order to simplify the presentation, we assume that the minimum meeting duration \(\mu (\mu \leq m_q)\) is known before the preprocessing, and set \(m_q = \mu\), 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 in the form \((t, object_id, location)\), ordered by the location reporting time \(t\). We start the preprocessing by dividing the time domain into a non-overlapping time intervals of equal duration (time blocks). Each time block (denoted as \(B_k\)) 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 \(C\). How to find an optimal value of \(C\) will be discussed in Section [VII].

For each time block, during the preprocessing, the following steps have to be completed: (i) computing candidate contacts, (ii) verifying contacts (performed for 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 step (iv), which is the most challenging step of preprocessing.

During the preprocessing, information regarding each object \(O_i\) is saved in a data structure named \(\text{objectRecord}(O_i)\), which is created at the beginning of each time block \(B_k\) and deleted after all the needed information is written on the disk at the end of \(B_k\). \(\text{objectRecord}(O_i)\) has the following fields: \text{Object_id}, \text{Cell_id} (the object’s placement in the grid with side \(H\) when it was first seen during \(B_k\)), \text{ContactsRec} (a list of the contacts of \(O_i\) during \(B_k\)), \text{MeetingsRec} (a list of meetings of \(O_i\) during \(B_k\)). The grid side \(H\) is another parameter (in addition to the contraction parameter \(C\)), which needs to be optimized. We will discuss this question in Section [VII]. In addition, for each time block we maintain a hashing scheme, that enables

### TABLE I: Notation used in the paper

| Notation | Definition |
|----------|------------|
| \(\Delta T\) | Duration between two consecutive time instants |
| \(\Delta t\) | Duration between two consecutive reporting times |
| \(m_q, \mu\) | Required meeting duration and minimum meeting duration |
| \(O_S, OT\) | A source and a target objects |
| \(O^{(\tau_j)}\) | Instance of object \(O_i\) at time \(\tau_j\) |
| \(\tau_0(O_i)\) | Earliest time when object \(O_i\) was reached |
| \(d, r\) | Transfer decay and portion of transferred information |
| \(h, h_{max}\) | Actual and maximum allowed number of hops (transfers) |
| \(\nu\) | Threshold weight |
| \(g_{w(h)}\) | Actual weight of an item after \(h\) transfers |
| \(f_{w(O_i^{(\tau_j)})}\) | Weight, assigned to an item carried by \(O_i^{(\tau_j)}\) considering \(\nu\) |
| \(F_w(O_i^{(\tau_j)})\) | Assigned aggregate weight of all items carried by \(O_i^{(\tau_j)}\) |
| \(B_k, I_k\) | Time block & that spans time interval \(I_k\) |
| \(C, H\) | Contraction parameter and grid resolution |
to access each object’s information by the object’s id.

A. Computing Contacts and Finding Meetings

Two objects \(O_i\) and \(O_j\) are candidate contacts at reporting time \(t_k\) if the distance between them at that time is no greater than candidate contact distance \(d_{cc} = 2d_{max} + d_{cont}\) (where \(d_{max}\) is the largest distance that can be covered by any object during \(\Delta t\)). Candidate contacts can potentially have a contact between \(t_k\) and \(t_{k+1}\). To force all candidate contacts of a given object \(O_i\), to be in the same or neighboring with \(O_i\)’s cells, at each \(t_k\) we partition the area covered by the dataset into cells with side \(d_{cc}\). Now, to find all candidate contacts of object \(O_i\), we only need to compute the (Euclidean) distance between \(O_i\) and objects in the same and neighboring cells.

Using our assumption that between consecutive reporting times objects move linearly, at \(t_{k+1}\), we can verify if there were indeed any contacts between each pair of candidate contacts during the time interval \([t_k, t_{k+1}]\). 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 of \((O_i)\).

At the end of each time block, a meeting duration \(m\) 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 (meetings that either start at the beginning or finish at the end of \(B_k\)) 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, during the preprocessing, for each object \(O_i\), we precompute all objects that are \((\mu, d)\)-reachable from \(O_i\) during \(B_k\). Here we are facing a challenge: to find, which objects can be \((\mu, d)\)-reached by \(O_i\), we need to know the transfer decay \(d\) and weight threshold \(\nu\), which are assumed to be unknown at the preprocessing time.

To overcome an issue of unknown \(d\) and \(\nu\), we turn our problem of reachability with decay into hop-reachability problem. Recall that 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, the allowed number of transfers (or hops) \(h\) for a successful meeting should satisfy the following inequality:

\[
h \leq \log_p \frac{\nu}{w},
\]

and finally

\[
h_{max} = \lceil \log_p \frac{\nu}{w} \rceil. \tag{6}
\]

Now the problem can be stated as follows: for each object \(O_i\), compute all objects, that are \((\mu, h_{max})\)-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, where an imaginary vertical line sweeps the \(xy\)-plane, left-to-right, stopping at points where information needs to be analyzed. In our case, the \(x\)-dimension is the time-dimension, and \(y\)-dimension is the order in which the meetings are discovered.

We demonstrate how the algorithm works on the Example in Fig. 3 and later provide a pseudo-code and detailed explanation. Consider the data in the table (a1). 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 that are 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 object \(O_1\) is reached (with \(h_{min} = 0\), which is recorded in table (b1)). During the given time block, \(O_1\) has only one meeting, \(\langle O_1, O_3, [0, 3] \rangle\) which is placed on the plane (a2). As a result of this meeting, object...
$O_3$ is reached at time $\tau = 2$, with the minimum hop-value $h_{\text{min}} = 1$, which is recorded in the table (b2). The sweep line moves to the time $\tau = 2$ - time, when object $O_3$ 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: $< O_3, O_2, [1, 5] >$. Even though it begins at $\tau = 1$, the retransmission does not start until $\tau = 2$, since only at this time $O_3$ becomes reached. As a result of these two meeting with object $O_3$, $O_2$ and $O_4$ become reached at $\tau = 4$ and $\tau = 7$ respectively, with $h_{\text{min}} = 2$ ((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 $h_{\text{min}}$ value may decrease with time. For example, object $O_4$ was reached by $O_2$ with $h_{\text{min}} = 3$ at $\tau = 6$ ((a4), (b4)), however as a result of the meeting with object $O_3$, its $h_{\text{min}}$ value went down to $h_{\text{min}} = 2$ at $\tau = 7$ ((a3), (b3)).

\begin{algorithm}
\caption{Reach($h_{\text{min}}$)}
1: Input: $O_S$
2: procedure UpdateHmin($O_i, r_i, \tau_i, h$)
3: for each $t_k \in \tau_i$ do $h_{\text{min}}(O_i^{t_k}) = h$
4: for each $O_i$ do
5: $\tau_R(O_i) = \infty$
6: UpdateHmin($O_i, r_i, \tau_{\text{end}}, \infty$) \triangleright $\tau_0$ and $\tau_{\text{end}}$ are the first and last time units of a block.
7: procedure ReachHop($O_S$)
8: $\tau_{\text{time}} = 0$, $\tau_R(O_S) = 0$, UpdateHmin($O_S, r_0, \tau_{\text{end}}, 0$), $S_{PQ} = \{O_S\}$
9: $S_{\text{ReachHop}} = \emptyset$
10: while ($S_{PQ} \neq \{\emptyset\}$ and $\tau_{\text{time}} \leq \tau_{\text{end}}$) do
11: $O_i = \text{ExtractMin}(S_{PQ})$
12: $S_{\text{ReachHop}} = S_{\text{ReachHop}} \cup O_i, \tau_{\text{time}} = \tau_R(O_i)$
13: for each $O_j$ that had a valid meeting with $O_i$ do
14: if $O_j \notin S_{\text{ReachHop}}$ then
15: $\tau_{\text{new}}(O_j) = \infty$
16: while $\tau_{\text{new}}(O_j) \geq \tau_R(O_j)$ do
17: read next meeting $M_{ij} = O_i, O_j, [\tau_i, \tau_j] >$
18: $\tau_{\text{new}}(O_j)$ = compute $\tau_{\text{new}}(O_j)$
19: if $\tau_{\text{time}}(O_j) > \tau_R(O_j)$ then
20: Update ($S_{PQ}, O_j$), $h = h_{\text{min}}(O_j^{\text{time}}) + 1$
21: UpdateHmin($O_j, \tau_{\text{new}}, \tau_R(O_j - 1, h)$)
22: if ($M_{ij}$ = last meeting $< O_i, O_j >$ in $B_k$) then
23: $\tau_{\text{new}}(O_i) = -1$
24: return $S_{\text{ReachHop}}$
\end{algorithm}

The process for computing all objects that are ($h_{\text{min}}$)-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 $S_{\text{ReachHop}}$ set keeps all objects for which all $h_{\text{min}}$ values as well as the earliest reached time had been computed and finalized. Those objects that were found to be reached, but not in $S_{\text{ReachHop}}$ 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 $S_{\text{ReachHop}}$ (lines 10, 11). At this time, all meetings of objects that can be reached by $O_i$ (but not in $S_{\text{ReachHop}}$) are analyzed (lines 13 - 23). As a result, both $\tau_R(O_j)$ (and their priority in $S_{PQ}$) as well as their $h_{\text{min}}$ values can be changed (lines 19 and 21). This algorithm has to be performed for each object of the dataset that is active during the given time block.

![Fig. 4: Two-level index on files Meetings and Reached(Hop).](image)

C. Index Construction

The index structure of RICCDelay is similar to the one of RICCMet 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). The records in Meetings Index are organized and follow the order of time blocks. Each record contains an object id and a pointer to the page with the first record for this object (for the given time block) in file Meetings. In the Reached Index, each record consists of an object id and a pointer to the page with the first record for this object for the given time block in file Reached(Hop). Each record in Time Block Index points to the beginning of a time block in Meetings Index and Reached Index.

V. Reachability Queries with Decay: 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 $m_d = \mu$.) First, using equation (6), we rewrite the problem as hop-reachability problem, replacing $w, d$, and $\nu$ from $Q_{md}$ with $h_{\text{max}}$. The new query can be written as $Q_{mk}: \{O_S, O_T, h_{\text{max}}, [\tau_s, \tau_f], \mu\}$.

The processing of $Q_{mh}$ starts from computing the time blocks $B_0, . . . , B_f$ 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 $S'_{\text{reached}}$ 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 object $O_S$. Recall that in Reached(Hop) every object $O_i$ that can be reached by object $O_s$ is recorded together with the smallest number of transfers $h_{\text{min}}$ that is required for $O_i$ to reach $O_s$. Thus during the query processing, an object $O_j$ cannot be considered as reached during the block $B_k$ unless $h_{\text{min}}(O_j^{B_k}) \leq h_{\text{max}}$ (where $h_{\text{min}}(O_j^{B_k})$ is the value $h_{\text{min}}$ of
object $O_j$ at the end of $B_k$). So, each object $O_j$ that was found to be reached by $O_S$ (a companion of $O_S$), is added to $S_{\text{reached}}'$ along with the corresponding number of hops $h_{\text{min}}$, provided that $h_{\text{min}}(O^{B_m}) \leq h_{\text{max}}$. Next, we proceed to block $B_{k+1}$. This time, retrieving all the companions of each object from $S'_{\text{reached}}$ and updating it by either adding new objects or adjusting the $h_{\text{min}}$ value for the objects that are already in the set. Such adjustment may be needed if, for some object $O_i \in S'_{\text{reached}}, h_{\text{min}}(O^{B_k}) > h_{\text{min}}(O^{B_{k+1}})$.

The process continues until $O_T$ is added to $S'_{\text{reached}}$ while reading some block $B_i(i < f)$ or the last block $B_f$ is reached.

If at the end of processing $B_f$, $S'_{\text{reached}}$ does not contain the target $O_T$, the query processing can be aborted, otherwise it moves to the file $\text{Meetings}$. Now 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 objects $O_i$ 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 one if needed. Also, if object $O_i$ was reached by the source object $O_S$ during the block $B_k$ with $h_{\text{min}}(O^{B_k}) = h_1$, and in a later block $B_m$, object $O_j$ was reached by $O_i$, within $h_2$ hops, $h_{\text{min}}(O^{B_m}) = h_1 + h_2$. Object $O_j$ is considered to be reached by $O_S$ if $h_{\text{min}}(O^{B_m}) \leq h_{\text{max}}$.

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

VI. TOP-K REACHABILITY: QUERY PROCESSING

To process top-k reachability queries efficiently, we will use the preprocessed data and index structure from RIC-Cdeca, described in the previous section. For that reason, we named our top-k reachability query processing algorithm $\text{RICTopK}$. The top-k query $Q_{\text{topK}}$ is issued in the form $\{S, W, D, \{r_s, r_f\}, \mu, \nu, k\}$, where $S = \{O_{S_1}, O_{S_2}, ..., O_{S_s}\}$, $W = \{w_1, w_2, ..., w_q\}$, and $D = \{d_1, d_2, ..., d_q\}$ are the sets of source objects, weights, and decays respectively. To make use of the precomputed data from RIC-Cdeca, the top-k reachability with decay problem has to be translated into top-k hop-reachability problem. Hence, for each source object $O_{S_i} \in S$, we compute $h_{\text{max}(r)}$ by applying inequality (6) to each triple $(O_{S_i}, w_r, d_r)$ as follows:

$$h_{\text{max}(r)} = \log_{p_r} \frac{\nu}{w_r},$$

where $p_r = 1 - d_r$, and $r = \{1, 2, ..., q\}$.

Now each top-k query can be thought of as written in the form $\{S, H_{\text{hop}}, \{r_s, r_f\}, \mu, \nu, k\}$, where $H_{\text{hop}} = \{h_{\text{max}(1)}, h_{\text{max}(2)}, ..., h_{\text{max}(q)}\}$. Note that the top-k query processing is the extension of the reachability with decay query processing algorithm, and thus we will avoid repeating some details concerning the use of the index structure during the query processing that were described earlier.

First, the set of $\text{Top-k Candidates}$ is initialized by adding to it all source objects. We start reading file $\text{Reached}(Hop)$ from time block $B_s$, checking all records for each source object from set $S$ (in order of their appearance in the file). Once an object, that was reached by at least one source, is discovered, it is added to $\text{Top-k Candidates}$. For each top-k candidate $O_i$, we keep the information about the source object(s), that it was reached by and $h_{\text{min}(r)}$ required to transfer information from each source to $O_i$. The search continues in this manner until time block $B_f$ is processed, after which the weight of each object from $\text{Top-k Candidates}$ is computed. Note, that this is not the actual weight $F_w$ of an object, but the maximum weight $F_{w_{\text{max}}}$ that this object may receive.

Next, the query processing moves to the file $\text{Meetings}$. Here, the algorithm maintains two structures: $\text{Top-k Candidates}$ and $\text{Top-k}$, that have to be updated at the end of each block. $\text{Top-k Candidates}$ contains: (i) the ids of all reached objects, (ii) their corresponding maximum weights $F_{w_{\text{max}}}$, as well as (iii) the current weight $F_w$ of each candidate top-k object. At the beginning, the weight $F_w$ of each source object $O_{S_i}$ is set to its initial weight $w_r$, while the rest of the objects’ weights $F_w$ are set to 0. Top-k is initialized by adding to it $k$ source objects from set $S$ with the top k weights; the weight $F_w$ of each top-k object is recorded as well.

Let us denote the lowest weight $F_{w_{\text{min}}}$ among the objects in $\text{Top-k}$ as $F_{w_{\text{min}}}$. If Top-k contains $k$ objects, and the object with the smallest value carries weight $F_{w_{\text{min}}}$, any object $O_i$, such that $F_{w_{\text{max}}}(O_i) < F_{w_{\text{min}}}$, cannot be among the top-k.

In file $\text{Meetings}$, the query processing starts from time block $B_s$. After one time block is processed, the aggregate weight $F_w$ of objects from $\text{Top-k Candidates}$ that were involved in some transfers, may increase, and has to be updated. This may lead to changes in Top-k. After Top-k and $F_{w_{\text{min}}}$ are updated, all objects $O_i$ from Top-k Candidates, such that $F_{w_{\text{max}}}(O_i) < F_{w_{\text{min}}}$, can be removed from the set of candidates. When the work on $B_s$ is completed, we proceed to the next block. This process continues until either the last time block $B_f$ of the query is reached or the size of $\text{Top-k Candidates}$ is reduced to the size of Top-k. The final state of Top-k answers the query.

For example, consider the top-k query with three source object $O_1$, $O_2$, and $O_7$, whose corresponding weights are 3, 4, and 3. Suppose, the query interval $[\tau_s, \tau_f]$ is contained in time blocks $B_1 - B_5$. Fig.[5] illustrates the example. Fig.(a1)-(a4) show the time blocks in files $\text{Reached}(Hop)$ and $\text{Meetings}$ that are being processed at the given stage, tables (b1)-(b4) display the $\text{Top-k Candidates}$ with their maximum possible aggregate weights $F_{w_{\text{max}}}$ and current aggregate weights $F_w$. The last column of tables, (c1)-(c4), keeps track of the current state of the Top-k set. Both, Top-k Candidates and Top-k are created after $\text{Reached}(Hop)$ is processed and updated after the corresponding time block of file $\text{Meetings}$ is processed.

The query answering begins in $\text{Reached}(Hop)$. The relevant data is read from blocks $B_1 - B_5$, and by the end of $B_5$, the supersets of all objects that can be reached by the source object is identified. These objects are $\text{Top-k Candidates}$. They are recorded in the Top-k Candidates table, together with their maximum possible aggregate weight $F_{w_{\text{max}}}$ (b1). Since at this stage the aggregate weight $F_w$ is known only for the source objects, the objects $O_1$, $O_2$, and $O_7$ are placed in the Top-k (c1). The query processing moves to $B_1$ in file $\text{Meetings}$ (a2). At the end of $B_1$, the aggregate weight of some objects $F_w$
is updated, and thus both, Top-k Candidates and Top-k are updated as well ((b2), (c2)). We notice that $F_w\text{min}(O_2) = 6$. Thus all objects $O_i$ with $F_{\text{max}}(O_i) < 6$ can be removed from the set of candidates. (Such objects are shown in gray in (b3) and (b4).) The next block is $B_2$ (a3), and after updating both tables ((b3) and (c3)), we exclude $O_3$ and $O_5$ from further consideration. After processing $B_3$, we remove $O_1$ from Top-k Candidates. Even though, the query interval ends only in $B_3$, the query can be suspended as the size of Top-k is reduced to the size of Top-k.

VII. EXPERIMENTAL EVALUATION

We proceed with the results of the experimental evaluation of RICCdecay and RICCtopK. Since there are no other algorithms for processing spatiotemporal reachability queries with decay, we compare against a modified version of RICCmeetMin [3] that enables it to answer such queries. All experiments were performed on a system running Linux 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.

A. Datasets

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 [37], which generates traces of objects, moving on real road networks. For the underlying network we used the San Francisco Bay road network, which covers an area of about 30000 $km^2$. These sets contain information about 1000, 2000, and 4000 vehicles respectively (denoted as $MV_1$, $MV_2$, and $MV_3$). The location of each vehicle is recorded every $\Delta t = 5$ seconds during 4 months, which results in 2,040,000 records for each object. The size of each dataset (in GB) appears in Table III. For the experiments on these sets, $d_{\text{cont}} = 100$ meters (for a (class 1) Bluetooth connection).

| Dataset | $MV_1$ | $MV_2$ | $MV_3$ | RW_1 | RW_2 | RW_3 |
|---------|--------|--------|--------|------|------|------|
| Size of Dataset (GB) | 54 | 107 | 213 | 97 | 194 | 487 |

We set $d_{\text{cont}} = 10$ meters (to identify physical contacts or contacts in the range of a Bluetooth-enabled devices).

B. Experimental Results

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.

B. Parameter Optimization

The values of the contraction parameter $\mathcal{C}$ and the grid resolution $\mathcal{H}$, that are used for the preprocessing, depend on the datasets. For each dataset, the parameters $\mathcal{C}$ and $\mathcal{H}$ were tuned on the 5% subset as follows. We performed the preprocessing of each subset for different values of $(\mathcal{C}, \mathcal{H})$, and tested the performance of RICCdecay on a set of 200 queries. The length of each query was picked uniformly at random 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 carried by the reached object and thus presents less interest). The parameters $\mathcal{C}$ and $\mathcal{H}$ were varied as follows: grid resolution $\mathcal{H}$ - from 500 to 4000 meters for MV datasets, and from 250 to 2000 m for RW datasets; contraction parameter $\mathcal{C}$ - from 0.5 to 30 min. For each dataset, the pair $(\mathcal{C}, \mathcal{H})$ that minimized the number of I/Os was used for the rest of the experiments. For example, for $MV_1$ we used $\mathcal{C} = 14$ min and $\mathcal{H} = 20000$ m, while for $RW_4$ we used $C = 2$ min and $H = 500$ m.

C. Preprocessing Space and Time

The sizes of the auxiliary files and the index sizes for RICCmeetMin and RICCdecay appear in Table II. RICCdecay uses about 13.5% more space compare to RICCmeetMin since it records more information into the file (reached(Hop)). For each reached object, in addition to its id, it saves its hop value.) The time needed to preprocess one hour of data for RICCdecay ranges from 14 sec for $MV_1$ to 91 min for $RW_4$. For comparison, the preprocessing time for RICCmeetMin ranges from 13 sec for $MV_1$ to 56 min for $RW_4$.

D. 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 $\mu$ 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.

Increasing the Maximum Allowed Number of Transfers.

In this set of experiments, we analyze the impact of $h_{\max}$ on the performance of the RICCdecay, and compare RICCdecay with RICCmeetMin. (The last was modified to enable it to answer reachability queries with decay.) 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. 7 (a1 – b3). RICCdecay accesses from 1.8 (for $MV_2$ dataset) to 11.5 (for $RW_4$ dataset) times less pages than RICCmeetMin. The biggest advantage of RICCdecay is achieved for $h_{\max} = 1$ for all datasets, and in general, the smaller the $h_{\max}$, the better is the performance of the RICCdecay algorithm. When answering a query $Q_{\text{mph}}$, it reads file Reached(Hop) first. File Meetings needs to be read only if during traversing file Reached(Hop), the target object appears among the objects, reached by the source (i.e. if $O_T \in S_{\text{Reached}}$). However, $S_{\text{Reached}}'$ is a superset of the set of objects that can be reached by $O_S$ during the query interval $I$. We say that a query is pruned, if it aborts after reading file Reached(Hop) because of not finding the target among the reached objects. By precomputing the hop value of each reached object, Reached(Hop) gives more accurate information, than RICCmeet, which reduces the size of $S_{\text{Reached}}$. The smaller the $h_{\max}$, the less objects are in $S_{\text{Reached}}'$, and thus the higher percent of queries can be pruned.

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. 7. 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.

Top-K Reachability Queries.

All the queries considered in this section until now were one-to-one queries: they had one source and one target object. Top-k queries that we described in Section III may have more than one source and one target objects. Multiple sources lead to the increase in the search space, while multiple undefined targets prohibit from the early query suspension. In addition, the need to calculate and compare the aggregate weights of the reached objects makes it impossible to prune a query (suspend it after just searching the file Reached(Hop)).
For each of our top-k experiments, we used sets of 100 queries, where query length was 3500 sec for MV datasets and 4200 sec for RW datasets. The number of source objects was set $4: S = \{O_{S_1}, O_{S_2}, O_{S_3}, O_{S_4}\}$, and each weight was assigned a value of 1. Further, $D = \{0.10, 0.15, 0.20, 0.25\}$, $\nu = 0.6$, and $k$ was randomly picked from 4 to 20. The area covered by each dataset is very large, so to force objects to be reached by several sources, for each query, we picked source objects from the same cell (with the side equal to $d_{c_0}$) at the beginning of the query interval. The results (see Fig. [8]) indicate that for top-k queries RICC meetMin accesses in average about 37% more pages than RICC topK for the MV, and about 30% more pages for RW datasets. The advantage of RICC topK owes to both, the RICC decay index, and RICC topK query processing. Information from the preprocessing allows for computing the maximum possible aggregate score $F_{\text{max}}$ using information from file Reached(Hop), while RICC topK reduces the number of objects that have to be accessed when the query reads the file Meetings.

Fig. 8: Top-k reachability queries

VIII. CONCLUSIONS

We presented two novel reachability problems: reachability with transfer decay and top-k reachability with transfer decay. To process these queries efficiently, we designed two new algorithms: RICC decay and RICC topK. The RICC meetMin algorithm [3] was modified to answer the same types of queries, and served as a benchmark. We tested our algorithms on six realistic datasets, varying query duration and the maximum allowed number of hops. The performance comparison showed that RICC decay and RICC topK can answer the new types of queries more efficiently than RICC meetMin.

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. Strzhemecky 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] E. Jin, N. Ruan, S. Dey, and J. Y. Xu, “Scrab: scaling reachability computation on large graphs,” in *ACM SIGMOD*, 2012, pp. 169–180.

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

[6] 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.

[7] 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.

[8] R. Jin, Y. Xiang, N. Ruan, and D. Fuhr, “3-hop: a high-compression indexing scheme for reachability query,” in *ACM SIGMOD*, 2009, pp. 813–826.

[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. Yıldırım, 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, “PreCh: 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] J. Tang, M. Musolesi, C. Mascolo, and V. Latora, “Characterising Temporal Distance and Reachability in Mobile and Online Social Networks,” in *ACM SIGCOMM Comp. Comm. Review*, vol. 40, no. 1, 2010.

[15] 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.

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

[17] G. Kollios, D. Gunopulos, and V. Tsotras, “On indexing moving objects,” in *ACM PODS*, 1999, pp. 261–272.

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

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

[20] M. Yiu, Y. Tao, and N. Mamoulis, “The bdual-tree: Indexing moving objects by space filling curves in dual space,” *VLDB J.*, vol. 17, no. 3, pp. 379–400, 2008.

[21] S. Chen, B. Ooi, K. Tan, and M. Nascimento, “St2b-tree: A self-tunable spatio-temporal b+-tree index for moving objects,” in *ACM SIGMOD*, 2008, pp. 29–42.

[22] J. M. Patel, Y. Chen, and V. P. Chakka, “Stripes: An efficient index for predicted trajectories,” in *ACM SIGMOD*, 2004, pp. 635–646.

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

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

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

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

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

[28] 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.

[29] H. Jeung, M. Yiu, X. Zhou, C. Jensen, and H. Shen, “Discovery of convoys in trajectory databases,” in *PVLDB*, vol. 1, no. 1, 2008, pp. 1068–1080.

[30] D. Wu, M. L. Yiu, G. Cong, and C. S. Jensen, “Joint top-k spatial keyword query processing,” *IEEE TKDE*, vol. 24, no. 10, 2012.

[31] D. Wu, M. L. Yiu, C. S. Jensen, and G. Cong, “Efficient continuously moving top-k spatial keyword query processing,” in *Data Engineering (ICDE), 2011 IEEE 27th Int. Conf. on.* IEEE, 2011, pp. 541–552.

[32] L. Chen, G. Cong, C. S. Jensen, and D. Wu, “Spatial keyword query processing: an experimental evaluation,” in *Proc. VLDB Endowment*, vol. 6, no. 3, 2013, pp. 217–228.

[33] J. Rocha-Junior, A. Vlachou, C. Doulkeridis, and K. Norvág, “Efficient processing of top-k spatial preference queries,” *Proc. VLDB*, vol. 4, no. 2, pp. 93–104, 2010.

[34] M. Attique, H. Cho, R. Jin, and T. Chung, “Top-k spatial preference queries in directed road networks,” *ISPRS Int. Journal of Geo-Inf.*, vol. 5, no. 10, p. 170, 2016.

[35] P. Ahmed, M. Hasan, A. Kashyap, V. Hristidis, and V. J. Tsotras, “Efficient computation of top-k frequent terms over spatio-temporal ranges,” in *Proceedings of the 2017 ACM ICMD*, 2017, pp. 1227–1241.

[36] A. Skovsgaard, D. Siddaaktas, and C. S. Jensen, “Scalable top-k spatio-temporal term querying,” in *Data Eng.*, IEEE, 2014, pp. 148–159.

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

[38] D. Maltz, “Dynamic source routing in ad hoc wireless networks,” *Mobile Computing*, vol. 353, no. 1, pp. 153–181, 1996.
This figure "fig1.png" is available in "png" format from:

http://arxiv.org/ps/2105.08312v1